VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZOVÁNÍ MAP KVALITY KŘEMÍKOVÝCH PLÁTŮ V REÁLNÉM ČASE

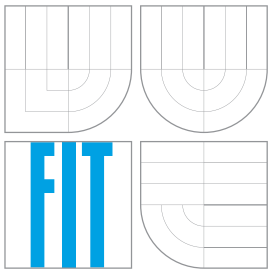
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MATĚJ MAREČEK

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZOVÁNÍ MAP KVALITY KŘEMÍKOVÝCH PLÁTŮ V REÁLNÉM ČASE

REAL-TIME DISPLAYING QUALITY MAPS OF SILICON WAFERS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MATĚJ MAREČEK

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. ADAM HEROUT, Ph.D.

BRNO 2013

Abstrakt

Cílem této bakalářské práce je navrhnout a implementovat softwarový systém, který dokáže v reálném čase sbírat data z průběhů měření křemíkových plátů. Může se jednat o desítky až stovky měření a data z těchto měření jsou uživatelům vykreslována (taktéž v reálném čase). Práce obsahuje popis procesu výroby a testování integrovaných obvodů. Následně pak návrh architektury systému aplikací, vnitřní architektury serveru a grafického uživatelského rozhraní klientské aplikace. V poslední části je ukázáno, jakým způsobem bylo implementováno vykreslování křemíkových plátů na platformě JavaFX 2.2 a hybridní vícevláknová architektura serveru.

Abstract

The goal of this bachelor's thesis is to design and implement a software system that can collect real-time data from measurements of silicon wafers. There could be tens or hundreds data sources of measurements and data from these measurements can be rendered (also in real time). This work contains a description of the process of manufacturing and testing of integrated circuits. Subsequently, there is description of system architecture design, interior architecture of real-time server and GUI of client application. In the last section, there is shown how rendering of silicon wafers was implemented on platform JavaFX 2.2 and also implementation of hybrid multi-threading server architecture.

Klíčová slova

Křemíkové desky, zobrazování v reálném čase, JavaFX 2.2, grafické uživatelské rozhraní, server, paralelismus

Keywords

Silicon wafers, real-time displaying, JavaFX 2.2, graphical user interface, server, parallelism

Citace

Matěj Mareček: Zobrazování map kvality křemíkových plátů v reálném čase, bakalářská práce, Brno, FIT VUT v Brně, 2013

Zobrazování map kvality křemíkových plátů v reálném čase

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. Ing. Adama Herouta, Ph.D. Informace k práci jsem čerpal ze zdrojů uvedených v seznamu literatury a od pana Matějky z firmy ON Semiconductor.

.....
Matěj Mareček
18. dubna 2013

Poděkování

Rád bych touto cestou poděkoval mému vedoucímu práce Doc. Ing. Adamu Heroutovi, Ph.D, který mě vedl k cíli, inspiroval a dělil se se mnou o své znalosti z oblasti návrhu uživatelských rozhraní. Také bych rád poděkoval kolegům z ON Semiconductor (především pánům: Jiřímu Kulhánkovi, Stanislavu Pobořilovi, Jaroslavu Bernkopfovi a Vojtěchu Kocurkovi), kteří se svou kritikou a nápady podepsali na finální podobě GUI. V neposlední řadě si můj dík zaslouží Vít Matějka, bez kterého by nebylo možno tuto práci prakticky realizovat a otestovat v reálných podmínkách.

Obsah

Kapitola 1

Úvod

Tato práce má za úkol řešit problémy, které vznikají, pokud je potřeba v reálném čase přenášet data z několika (i stovek) vzdálených zdrojů a zobrazovat je v klientské aplikaci. V tomto konkrétním případě se jedná o zobrazování map kvality křemíkových plátů v reálném čase, které vzniklo ve spolupráci s firmou ON Semiconductor.

Účelem následujícího textu je poskytnout čtenáři základní informace o procesu výroby integrovaných obvodů a jejich testování. Budou také popsány některé důležité aspekty tvorby grafických uživatelských rozhraní, které byly v rámci praktické části práci aplikovány. Na konkrétním příkladu bude ukázáno, jakým způsobem byl tvořen návrh systému aplikací a jejich architektura. U aplikace Mapper Dashboard budou popsána řešení a koncepty, které byly užity při tvorbě GUI.

Některá implementačně složitá, či zajímavá témata budou rozebrána v poslední části. Důraz bude kladen především na možnosti využití paralelismu, spolupráci vláken a zakomponování nových modulů do již existujícího software a jeho architektury. Mimo jiné zde najdete implementační řešení, kterým se podařilo úspěšně odstranit problémy, jež měla JavaFX 2.2 při vykreslování velkého množství dat.

V samotném závěru budou shrnuty dosažené výsledky (kolik probíhajících měření křemíkových plátů se podařilo vykreslovat v reálném čase a kde všude byl tento software nasazen do produkčního prostředí) a nastíněna určitá vize vývoje celého projektu ve střednědobém časovém horizontu.

Kapitola 2

Výroba integrovaných obvodů a úloha Real-time Mappingu v tomto procesu

První kapitola je spíše teoreticky orientovaná a čtenáři by měla poskytnou dostatek informací, které bude potřeba znát v dalších kapitolách.

V první části jsou uvedeny a konkretizovány pojmy vyskytující se dále v práci. Poté je popsán proces výroby integrovaných obvodů/čipů a také vysvětleno, jakým způsobem tato práce do procesu výroby zapadá a jaký je její účel.

Druhá část je poté věnována některým důležitým aspektům vývoje uživatelského rozhraní, které jsem se poté snažil aplikovat při tvorbě výsledného programu.

2.1 Základní pojmy a terminologie

Tato sekce se zabývá definicí pojmů, se kterými se bude v další části bakalářské práce pracovat. Popsání významu jednotlivých pojmů je důležité jednak z důvodu možné mnohoznačnosti (wafer může být nejen z polovodičového materiálu, ale také z čokolády) a také kvůli bližšímu upřesnění (jak jsou chápány ve firmě ON Semiconductor a tedy i v této práci).

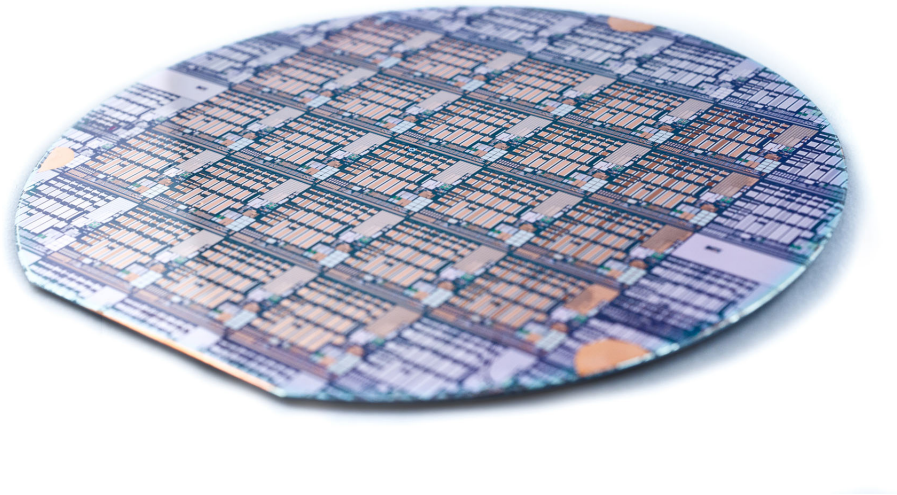
RTM

RTM je zkratka pro Real-time Mapping, respektive tuto bakalářskou práci. Jedná se o soubor programů, které mají za úkol dostat v reálném čase data z probíhajících měření až k uživateli.

Wafer

V oblasti výroby čipů a elektroniky obecně se pojmem wafer rozumí oválný plátek, který je většinou vyráběn z křemíku ve formě monokrystalu (jehož čistota bývá vyšší jak 99.999%), nebo z nějakého jiného polovodičového materiálu. Wafer slouží jako základ pro výrobu elektronických součástek (například různých čipů a mikro-obvodů) a mohou mít obecně různou velikost.

Velikost waferu se určuje jako jeho průměr a jako jednotky se většinou používají milimetry, případně palce. Obecně je snaha tento průměr zvyšovat, jelikož na větší ploše je



Obrázek 2.1: Obrázek waferu [16].

možno vyrobit více čipů a snížit tak náklady. V dnešní době se vyrábí wafery až o velikostech $450mm$ (často se označují jako 18 palcové) a tloušťce $925\mu m$.

Lot

Lotem je zde myšlena sada waferů stejného typu. Každý lot většinou obsahuje do 25 waferů.

BIN

BIN je celé nezáporné číslo o maximální teoretické hodnotě 255. Reprezentuje konečný výsledek testu, či testů, kterými byl podroben určitý čip/součástka. Číslo 0 většinou znamená, že testy dopadly v pořádku a ostatní čísla indikují nějaký typ chyby.

Barvy jednotlivých BINů jsou v ON Semiconductor definovány standardem. Čísla 0 až 16 mají každé různou barvu. Ta je ve standardu popsána pomocí RGB. Pro BINy s vyššími čísly (17 až 255) se poté opakuje perioda barev pro BINy s hodnotou 1 až 16.

YIELD

Číslo od 0 do 100, které udává procentuální výtěžnost. YIELD 66 tedy znamená, že dvě třetiny změřených čipů prošly testy v pořádku a u zbývající třetiny k nějakému problému došlo. Cílem při výrobě čipů je mít co nejvyšší výtěžnost. U waferů, jejichž výroba je již dobře zvládnuta, se toto číslo pohybuje nad 90%.

Wafer prober

Zkráceně jen prober, je stroj, který se používá pro testování integrovaných obvodů. Pohybuje waferem a na hliníkové plošky jednotlivých čipů přikládá kartu (probe card) s měřícími hroty.

Tester

Je zařízení, které je propojeno dráty s měřícími hroty (zjednodušeně si jej můžeme představit jako voltmetr). Každý tester má v sobě obsažen řídící program, který provádí testování jednotlivých čipů. Tyto programy, respektive testy, mohou být různě sofistikované (může se jednat o jednotky až stovky testů). Detailní výsledky testů se ukládají do databáze a finální výsledek ve formě BINu je zaslán aplikaci Mapper.NET (ta je popsána v sekci 2.1.1).

2.1.1 Programy v RTM

Mapper.NET

Mapper.NET (zkráceně označován jako Mapper) je interní software vyvíjený ve společnosti ON Semiconductor, který slouží ke kontrolování proberu a testeru. Tato aplikace je vyvíjena pro platformu Microsoft .NET Framework a používá se na počítačích v čistých prostorách (tento počítač je propojen s proberem a testerem).

Činnost Mapperu spočívá v tom, že pošle proberu příkaz, aby provedl zakontaktování. Jakmile je tato operace hotova, tak Mapper pošle příkaz *„začni měření“* do testeru. Ten autonomně provede sérii testů a vrátí výsledek ve formě BINu.

Mapper Web service

Jedná se o webovou službu určenou především pro aplikaci Mapper.NET. V rámci této práce byla použita jako zdroj permanentních dat o Mapperech a jejich rozložení v čistých prostorách.

RTM server

Celým jménem *Real-time Mapping server* je serverová aplikace napsaná v rámci této bakalářské práce. Jejím účelem je přijímat data od zdrojů (Mapperů) a přeposílat je klientům (Mapper Dashboard).

Tento server je napsán pro platformu Java 1.7, přičemž detailnější informace o jeho fungování a implementaci jsou k nalezení v kapitolách 3.2 a 4.2.

Mapper Dashboard

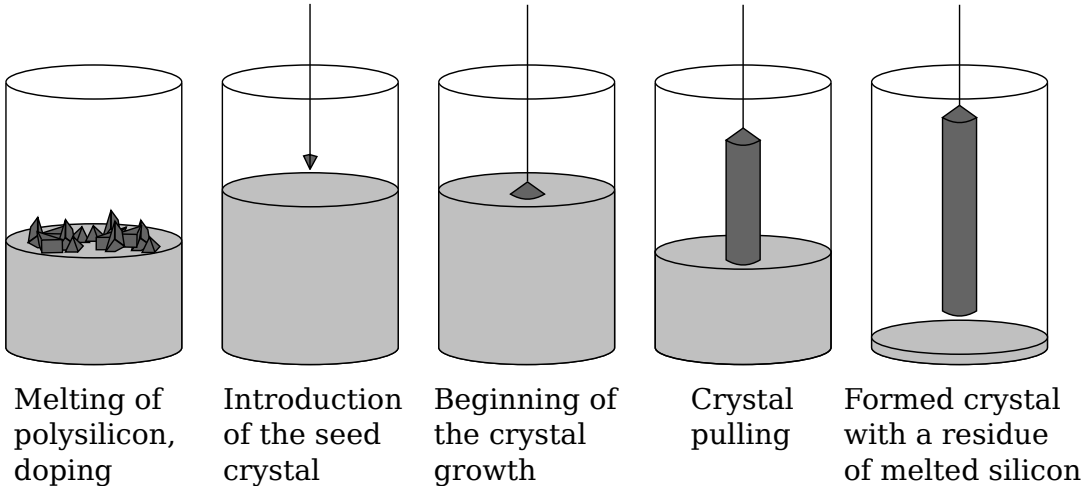
Mapper Dashboard je klientská aplikace naprogramovaná pro JavaFX 2.2 (Java 1.7) a slouží především k vizualizaci dat, které přijdou z Mapper Web service a RTM serveru. Jedná se o stěžejní část této práce, kvůli které byl modifikován Mapper.NET, Mapper Web service a vytvořen RTM server.

2.2 Výroba a testování integrovaných obvodů

2.2.1 Proces výroby integrovaných obvodů

Czochralského metoda růstu krystalu

Tato metoda růstu syntetických monokrystalů polovodičů byla popsána polským vědcem Janem Czochralským již v roce 1918. Od té doby byl tento proces značně vylepšen a stále se jedná o jeden z nejvíce oblíbených a používaných postupů, jak produkovat monokrystaly o velmi vysoké kvalitě.



Obrázek 2.2: Czochralského metoda [6].

Základ této metody je poměrně jednoduchý. Nejdříve se ve výhni (ta je anglicky označována jako *crucible* a tvořena většinou z křemene) roztaví křemík o vysoké kvalitě. Do roztaveného materiálu jsou dále přidávány atomy se 3, nebo 5 valenčními elektrony (jako je například bor nebo fosfor, jejichž koncentrace se pohybuje v rozmezí 10^{13} až 10^{16} atomů na centimetr krychlový), které dotují krystal. Poté se do této taveniny vloží zárodek (anglicky *seed crystal*), který rotuje v určitém směru (přibližně 10 až 100 otáček za minutu) a zároveň je pomalu vytahován nahoru směrem z výhně (rychlost vytahování je mezi 0.5 až 15 centimetrů za hodinu).

Teplota taveniny se pohybuje lehce na bodem tání křemíku (tedy mezi 1410 až 1420 stupni Celsia) a celý proces se odehrává v inertní atmosféře (nejčastěji se používá argon). Výsledný tvar monokrystalu je dán mnoha faktory, přičemž mezi nejdůležitější patří rychlost vytahování krystalu, teplota a rotace zárodku/výhně [6, 7].

Výroba čipů

Poté, co je monokrystal nařezán na pláty (wafery), vyleštěn a chemicky očištěn slabými kyselinami (je velice důležité mít co nejhladší povrch waferu), nastává samotná část procesu výroby čipů.

Tento proces výroby integrovaných obvodů je založen na vytváření jednotlivých vrstev (může jich být i několik desítek), přičemž každá z těchto vrstev má určitou masku. Tato maska se tvoří pomocí fotolitografie, a jakmile je hotová, tak se pomocí difuze přidají na nezamaskovaná místa určité příměsi. Takto se vytvářejí místa s polovodičovým materiálem typu N a typu P a vznikají tak PN přechody, které se chovají jako hradla a propouští elektrický proud určitým směrem.

Čipy jsou na waferu poskládány nejčastěji v maticovém tvaru, a jelikož jsou poměrně malé, tak se jich na jeden wafer vleze i několik tisíc.

2.2.2 Proces testování

Jelikož se u vyráběných integrovaných obvodů klade veliký důraz na miniaturizaci (snižuje se poté spotřeba čipu v provozu a zároveň se šetří na materiálu), stává se výroba čipů čím dál tím složitější a vzrůstají nároky na čistotu materiálu. Jediná drobná prachová částice, která dopadne, kam nemá, může způsobit nefunkčnost celé jednotky. Proto samotná výroba probíhá v takzvaných *čistých prostorách*, kde platí poměrně striktní omezení.

I přes tato striktní omezení a také vlivem dalších faktorů dochází k tomu, že některé čipy nefungují tak, jak by měly. A protože na správném fungování čipu mohou záviset lidské životy (zdravotnická a vojenská technika), případně může dojít k materiálním škodám, tak je potřeba provádět testování.

2.3 Úloha Real-time Mappingu v procesu výroby

Proces výroby integrovaných obvodů (viz kapitola 2.2.1) je poměrně komplexní záležitost a pokud v jeho průběhu dojde k nějaké chybě, tak výsledek může být nepoužitelný. Během výroby se sice provádí určité kontroly, ovšem až úplně na konci se ukáže, zdali je možno produkt, jehož výroba trvala týdny až měsíce, dát do prodeje.

Informace o výsledcích měření, jsou tedy velice zajímavé pro spoustu pracovníků firmy. A právě úkolem RTM je, aby tyto informace byly uživatelům (pracovníkům) co nejdříve dostupné a tím pomohly k řízení výrobního procesu. Bez Real-time Mappingu by se muselo čekat, až se celý wafer doměří (to může trvat i několik hodin) a výsledky se nahrají do globálních systémů.

Výhody RTM

- **Včasná reakce na problémy** - jelikož je průběh měření zobrazován uživatelům v reálném čase, tak je možné včas reagovat na případné problémy. V čistých prostorách jsou sice operátoři, kteří mají za úkol připravit vše pro měření a poté jej spustit, ovšem už nijak neřeší, jestli vycházejí nějaké neočekávané výsledky.

  Toto mají na starost test inženýři, kteří z průběhu měření dokáží odhadnout, jaká chyba nastala a provést příslušné kroky k nápravě. Například pokud měření probíhalo v pořádku a od určité doby začala přicházet data se špatnými výsledky měření, je docela pravděpodobné, že je problém se zakontaktováním testeru. Nebo pokud chodí pouze špatné výsledky, tak je možné, že operátor něco špatně nastavil.

  V takovýchto případech je dobré testování včas zastavit a chybu odstranit. Pokud by se totiž testovalo dále, dochází k opotřebování hliníkových plošek na testovaných součástkách. Pokud by k omylům došlo vícekrát, tak se tyto plošku mohou opotřebovat do té míry, že už není možno součástku změřit (a produkt, jehož funkčnost nebyla ověřena nelze prodávat).

- **Poslední stav Mapperu** - protože jsou součástí RTM i Mapper Web service, je možné získat poslední stav Mapperu i v případě, že není připojen k RTM serveru.

- **Možnost vidět více strojů najedou** - aplikace Mapper Dashboard disponuje funkcí sémantického zoomování, kdy je možno zoomováním, místo zvětšování/zmenšování grafiky, měnit množství informací. Tím je možno zobrazit i několik desítek Mapperů najednou (záleží ovšem na rozlišení monitoru a množství informací/podrobností, které chceme vidět).

- **Průběžný YIELD** - YEILD waferu se dostane do globálních systémů až po skončení měření. Pokud ovšem sledujeme průběh měření, tak zároveň dostáváme informaci o průběžném YIELDu, ze kterého se dá odhadnout výsledný YEILD (předpokládáme-li rovnoměrné rozložení chybovosti).

- **Virtuální přístup do čistých prostor** - RTM umožňuje částečně sledovat, co se děje v čistých prostorách, bez nutnosti fyzické prezence. Přístup do čistých prostor je totiž omezen (důvod je popsán v kapitole 2.2.2). Navíc příslušné osoby musí projít školením, musí se obléct atd. S RTM tento problém tedy částečně řeší.

2.3.1 Nasazení RTM

Real-time Mapping může být jednoduše nasazen v továrnách, které využívají Mapper.NET verze 3.5 a vyšší a také příslušné Mapper Web service. Jelikož je RTM server psán pro platformu Java 1.7, je možné jej spustit na serverech jak s Linuxem (tento typ se v ON Semiconductor vyskytuje nejčastěji), tak Windows. Mapper Dashboard je poté možno spustit na všech počítačích, které mají podporu JavyFX 2.2 a vyšší.

Místa, kde může být RTM nasazeno jsou následující:

- **Česká republika, Rožnov pod Radhoštěm** - již nasazeno v produkčním prostředí.

- **Japonsko, Niigata** - již nasazeno v produkčním prostředí.

- **Malajsie, Seremban** - zatím nenasazeno.

- **Filipíny, Manila** - zatím nenasazeno.

- **USA, Oregon, Gresham** - zatím nenasazeno.

2.3.2 Cílové skupiny uživatelů

Přestože je Mapper Dashboard dostupný prakticky všem zaměstnancům, tak existují určité skupiny lidí, kteří jej budou využívat častěji a určitým specifickým způsobem.

- **Test inženýři** - ti se starají o *„probe“* - tedy měření a inkování (inkování je starý způsob, kterým se značily součástky, které neprošly testováním).

- **Device inženýři** - starají se o výrobu jednotlivých zařízení. Sledují tedy celý proces výroby až po finální testy. V aplikaci Mapper Dashboard používají většinou detailní náhled na měřený wafer.

- **Operátoři** - aby nemuseli chodit po čistých prostorách a kontrolovat, zdali některé měření již skončilo, tak mohou sedět u počítače a sledovat jednotlivé průběhy na něm (viz obrázek 2.3).

- **Manažeři** - manažery zajímá většinou celkový přehled stavu pracoviště (pokud uvidí, že 7 z 10 strojů stojí už půl dne a nikdo s tím nic nedělá, mohou se zeptat příslušných pracovníků, proč tomu tak je) a průběhy měření některých prioritních lotů (může se jednat o nějaké testy, či urgentní zakázky).

Obrázek 2.3: Operátor ovládající Mapper.NET v čistých prostorách.

2.3.3 Zabezpečení

K datům z RTM serveru může mít přístup kdokoli, kdo je na firemní síti (nejedná se o tajné informace). Dokonce je možné přenášet data bez omezení mezi jednotlivými továrnami (například zaměstnanec v Rožnově p. R. se může dívat na průběh měření v Niigatě).

Další typy zabezpečení by se zavedly v případě, že aplikace Mapper Dashboard bude podporovat omezený vzdálený přístup k jednotlivým Mapperům (například vzdálené zastavení měření v případě, že zodpovědný zaměstnanec detekuje nějaký problém při testování).

2.4 Obecné zásady uživatelsky přívětivého GUI aplikované v Mapper Dashboard

Během tvorby aplikace Mapper Dashboard bylo potřeba nastudovat základní principy tvorby aplikací s grafickým uživatelským rozhraním. Teoretické informace o některých z těchto principů, které jsem považoval za důležité a pokusil se aplikovat v praxi, jsou popsány v této kapitole.

2.4.1 GUI standardy

V oblasti tvorby uživatelského rozhraní neexistuje něco jako oficiální, obecně platný a vše zahrnující standard pro jednotné rozhraní. On ani prakticky existovat nemůže. Za prvé nelze popsat GUI stejným exaktním způsobem jako například DNS protokol. A za druhé se v dnešní době používá hned několik druhů uživatelských prostředí. Na Linuxu je velice populární prostředí Gnome, na Windows 8 můžeme dokonce najít jak klasické desktopové Aero, tak mobilní Modern UI (Metro) a Apple má také vlastní prostředí. Každé z těchto prostředí (ať už je určeno primárně pro desktopové nebo mobilní aplikace) má svoje specifika, styl práce a pravidla, které by měl vývojář/designer GUI dodržovat.

Specifika daného prostředí, pro které vyvíjíme aplikaci, je dobré zohlednit právě kvůli uživatelům, kteří náš produkt budou používat. Ti si totiž na platformě, na které pracují, vypěstovali určité návyky a pokud je o ně aplikace připraví, tak jejich ochota takový software používat výrazně klesá a navíc se prodlužuje doba potřebná k jeho ovládnutí.

V psychologii jsou návyky definovány jako *„zautomatizované úkony dovedené do určitého stupně dokonalosti“* [5]. Návyky nás v zásadě motivují k tomu, abychom opakovali určitý vzorec chování a to často bez rozmyslu a reflexivně. Jako příklad lze uvést ranní čištění zubů [9]. Pokud má člověk tento návyk dostatečně zafixovaný, tak nemožnost vyčistit si po snídani zuby v dotyčném automaticky vyvolává nepříjemný pocit. Pokud bych měl tento příklad se zubní hygienou převést do světa počítačů, tak nemožnost připnout si aplikaci na Taskbar ve Windows nebo přemístit aplikaci z jednoho monitoru na druhý pomocí kláves Win+šipka, může vyvolávat u uživatele nepříjemné pocity a narušovat jeho práci.

Podobně tomu je, pokud nějaký návyk uživatele považujeme za zlozvyk a rádi bychom tento zlozvyk odstranili. Tento problém musel být řešen při vývoji aplikace Mapper Dashboard (v té době se jednalo o beta verzi), kde stačilo přesunout tlačítko, které schovávalo boční panel z pravé strany na levou (kde to měl uživatel blíže a nové umístění dávalo větší smysl).

Z těchto, ale i dalších důvodů je dobré dodržovat konvence a doporučení vydané pro dané prostředí, na kterém naše aplikace poběží (problém může nastat, pokud tvoříme multiplatformní GUI aplikaci).

Doporuční pro jednotlivé platformy

- **GNOME Accessibility Developers Guide** - https://developer.gnome.org
- **Windows User Experience Interaction Guidelines** - http://msdn.microsoft.com
- **OS X Human Interface Guidelines** - http://developer.apple.com
- **Android User Interface Guidelines** - http://developer.android.com

2.4.2 Prototypování na papír

Prototypování na papír je velice často používaná metoda při vývoji aplikací, jejichž rozmach nastal v 90. letech, kdy tuto metodu vývoje software začaly aktivněji používat firmy jako Microsoft a IBM. Cílem této metody je pomoci programátorům a designérům v co nejrychlejším a nejlevnějším návrhu software, který bude ve velké míře zohledňovat požadavky uživatelů a bude těžit z jejich zpětné vazby [10].

Výhodou prototypování je především jeho rychlost. Je možné udělat návrh vnitřní architektury softwaru nebo základní koncept uživatelského rozhraní bez toho, abychom napsali

jediný řádek kódu. Zároveň je zde možné návrh jednoduše změnit. Stačí vzít propisku, něco přeškrtnou, něco dopsat nebo dokreslit.

Poznatky z prototypování Mapper Dashboardu

Další výhodou tohoto způsobu prototypování je, že jej lze provádět v teamu. Stačí tužku s papírem vyměnit za tabuli a fix. Pokud totiž svůj návrh kreslíme na tabuli a zároveň s tím jej komentujeme a vysvětlujeme (vysvětlování nám pomáhá utřídit své vlastní myšlenky), tak ostatní členové teamu mají přímou možnost do tohoto procesu zasáhnout. To se hodí především v případě, že má nějaký člen teamu důležitou informaci, kterou ostatní nemají, nebo je nenapadlo ji aplikovat. V řešení určitého problému (jako je design GUI) se tak mohou provádět modifikace přímo v procesu návrhu. Jako vedlejší produkt tohoto způsobu práce si team synchronizuje myšlenky a tím se redukuje možnost případných budoucích nedorozumění.

Když se prototyp GUI ukazuje budoucímu uživateli, tak je výhodné dotyčnému člověku nesdělovat informace o tom, jak by se měl daný program používat, ale pouze sledovat jeho reakce a dělat si poznámky. Dotyčný tak odhalí svůj způsob uvažování a původní představy práce s daným programem. Až touto první fází projdeme, tak na základě získaných informací a zpětné vazby můžeme náš prototyp buď zcela přepracovat (pozor, větší změny v GUI se mohou dotknout i vnitřní architektury software), nebo upravit tak, aby uživateli více vyhovoval (tento druhý případ se týká spíše robustnějších řešení).

Pokud uživatel požaduje nějaké zásadní změny v GUI, je potřeba si s ním o tom detailně pohovořit a ujistit se, že plně rozumíme tomu, proč tyto změny chce a jaký mají význam. Pokud se nám na požadavcích uživatele něco nezdá, je dobré je konzultovat i s dalšími uživateli (nejlépe nějakým reprezentativním vzorkem) a vývojáři.

Prototypování architektury software a designu GUI nebývá dobré podcenit. Především pokud programátor nemá s vývojem dostatečné zkušenosti a tato fáze mu připadá, že zahrnuje zbytečně moc diskuzí a málo psaní kódu. Pokud se ovšem této etapě vývoje nevěnuje dostatečná pozornost, tak v dalších fázích životního cyklu software mohou nastat vážné problémy.

2.4.3 Odezva GUI

Jedním z atributů aplikace s GUI, která chce být oblíbená u uživatelů je doba její odezvy. V tomto případě nejde ani tak o to, jak dobře má aplikace optimalizované algoritmy a jestli požadovaný úkol zvládne za 10 nebo za 15 sekund. Důležité je, aby i v případech, kdy výpočet trvá delší dobu, bylo GUI responzivní a patřičným způsobem dávalo uživateli najevo, že se stále něco děje a program nezamrzl. Takovéto chování je kritické především u aplikací, které se ovládají dotekem. Není-li u takových aplikací GUI dostatečně responsivní, tak uživatelé prožívají pocity podrážděnosti, někdy i agrese (také se snižuje jejich produktivita) a odmítají aplikaci používat.

Studie *„System Response Time and User Satisfaction: An Experimental Study of Browser-based Applications“* [2], která se touto problematikou zabývá, dochází k následujícím závěrům:

- Spokojenost uživatelů s aplikací se snižuje s tím, jak roste doba odezvy.
- Okamžitá odezva aplikace také není nejlepší, jelikož uživatelé mají určitou frekvenci interakcí s rozhraním a navýšení této frekvence může vést z jejich strany k více chybám.

- Výzkum moderních hypertextových systémů ukázal, že uživatelé potřebují méně než jednu sekundu, pokud přechází mezi jednotlivými stránkami k tomu, aby se mohli volně pohybovat v informačním prostoru.
- Doba odezvy delší než 3 sekundy výrazně ovlivňuje spokojenost uživatelů s aplikací.
- Pokud je doba odezvy delší jak 12 sekund, tak roste pravděpodobnost toho, že se uživatelé rozhodnou danou aplikaci nepoužívat.
- Pomalá doba odezvy vadí všem uživatelům přibližně stejně bez ohledu na jejich zkušenosti. Respektive studie v této oblasti nenašla žádný signifikantní vztah, který by dokazoval opak.

Pokud bychom měli zobecnit výsledky výše zmíněné studie [2], nebo například práci *„Response Times: The 3 Important Limits“* [11], případně doporučení vydaná pro Gnome [15], dojdeme k následujícím závěrům:

- **50-150 milisekund** - takovouto reakční dobu by měly mít úkony jako je klik myši, pohyb kurzoru myši, změna velikosti okna atp. Uživatel by měl mít u těchto akcí pocit, že systém reaguje okamžitě.
- **1 sekunda** - do jedné sekundy by měly být provedeny časté a jednoduché akce. Uživatel si sice všimne určitého zpoždění a cítí, že počítač něco zpracovává, ovšem jeho myšlenkový proud nebude přerušen. Jako příklad je možno uvést řazení dat v tabulce. Pokud program pracuje s velkým množstvím dat a nedokáže je seřadit do jedné sekundy, myšlenkové pochody uživatele jsou narušeny a je potřeba mu dát patřičným způsobem najevo, že program pracuje (například změnit kurzor myši na přesýpací hodiny).
- **2-4 sekundy** - do tohoto časového rozpětí spadají výpočetně náročnější úkony (například výše zmíněná práce s rozsáhlejší tabulkou). Uživatel už ztrácí pocit kontinuity práce.
- **8-12 sekund** - mezi 8 až 12 sekundami jsou uživatelé většinou schopni udržet pozornost. Určitě je ale potřeba dát uživatelům najevo, že se stále pracuje a aplikace nezamrzla.
- **10 a více sekund** - uživatelé přestávají věnovat aplikaci pozornost a začínají dělat jinou (více či méně užitečnější) činnost jako je například kontrola emailu, Facebooku atp. V tomto případě už nestačí změnit kurzor myši na přesýpací hodiny/*progress circle*, ale je třeba ukázat uživateli něco, jako je procentuální stav činnosti aplikace (nejlépe ještě s odhadovaným zbývajícím časem). V případech, kdy není možné určit stav činnosti v procentech, je dobré uživateli ukázat alespoň nějaké absolutní hodnoty (například počet již stažených informací ze sítě atd.). Také by měl mít uživatel možnost dlouho provádějící úkoly ukončit jiným způsobem než vypnutím aplikace nebo zrušením procesu/vlákna přímo v operačním systému.

Kromě používání komponent jako je *progress bar* nebo *progress circle* je důležitý také jejich vzhled a chování, kterým u uživatele můžeme vyvolat dojem, že aplikace pracuje rychle (i když realita může být jiná). Studie *„Faster progress bars“* [3] ukazuje, že čekání se jeví uživateli kratší, má-li *progress bar* vlnitou výplň a pohybuje se směrem dozadu.

Tato iluze je způsobena tím, že naše vnímání pohybu je relativní ve vztahu ke grafickému kontextu. Pohybování vln v opačném směru nežli se pohybuje indikátor měření, vytváří iluzi zvýšené rychlosti, což může pozměnit naše vnímání doby trvání.

Pokud pro indikaci činnosti programu použijeme *progress circle* komponentu, tak můžeme k vytvoření iluze rychleji ubíhajícího času zvýšit rychlost otáčení (pulzů). Tento princip je podobný jako u hudby, kde se tak určuje tempo. Zatímco 4 minutová pomalá písnička může pocitově trvat 7 minut, tak rychlé basy a rytmus mohou v posluchači vyvolat zrychlené vnímaní času.

Studie *„Rethinking the progress bar“* [4] ukazuje, že jednoduchou možností, jak zpříjemnit a zkrátit uživateli čekaní, je vyhnout se situacím, kdy *progress bar* ukazuje 99% hotovo a zbylé jedno procento přitom trvá déle, než předcházející. Výzkumy v této oblasti ukazují, že uživatelé jsou ochotnější čekat spíše v začátcích než na konci prováděné úlohy. Je tedy lepší nechat uživatele čekat ve 20 procentech výpočtu než v 90 a více[1].

2.4.4 Konzistence GUI

Konzistence grafického uživatelského rozhraní je jeden z atributů, který dopomáhá novému uživateli bez zkušeností s daným programem k jeho rychlejšímu ovládnutí. Kromě konzistence GUI s doporučeními pro konkrétní operačním systémem/platformou (tato doporučení nemusíme vždy slepě dodržovat, pokud nám z nějakého důvodu nevyhovují) popsaných v části 2.4.1 je důležité, aby byla zachována také konzistence designu a práce s danou aplikaci jako celkem.

První a nejviditelnější pravidlo konzistence GUI je pro uživatele přívětivý design a logické rozložení kostry programu. Ani design, ani základní kostra programu by se v průběhu práce neměly nijak zásadně měnit[2]. Pokud například uživateli během jeho práce schováme panel nástrojů a místo toho mu zobrazíme klasické menu, navíc třeba vertikálně členěné, tak by se měl daný programátor/designer důkladně zamyslet, zdali je vůbec základní koncepce jeho GUI vhodná pro účely vyvíjené aplikace. Dobrým zvykem bývá, že každá část rozhraní má svůj neměnný účel (pokud to je možné). Pokud se tedy rozhodneme, že naše aplikace nebude mít klasické globální menu, ale místo něj použijeme postranní panel, kde budeme kontextově vkládat různá nastavení a ovládací prvky (tento přístup byl užit v aplikaci Mapper Dashboard a je detailněji popsán v kapitole 3.3.3), tak bychom to tak měli dodržet na většině stavů programu. Cílem takovéto druhu konzistence je, aby se uživatel dokázal v programu orientovat a aby pochopil význam jeho různých částí.

Máme-li konzistentní design a uživatel se v naší aplikaci orientuje, můžeme sjednotit chování jednotlivých funkcí programu, které vykonávají podobné činnosti. Jako dobrý příklad na demonstraci nám může posloužit textový editor podobný programu Writer z LibreOffice a jeho dvě funkce: *„udělej text tučný“* a *„podtrhni text“*. Obě funkce dělají podobné věci (mají nějak zvýraznit text). Ovšem představme si, že by se tučného textu dosahovalo tak, že první vybereme text a poté klikneme na příslušné tlačítko a pokud bychom chtěli text podtrhnout, tak první budeme muset kliknout na příslušné tlačítko a poté vybrat text, který má být podtržený. Jakkoli se tento příklad může zdát nereálný, tak tomu tak není, jelikož na jednom projektu může pracovat více vývojářů a každý z nich dělat na jiné funkcionalitě [12].

[1] Tento způsob lze použít v případech, kdy se špatně odhaduje, kolik procent práce je již hotovo. Není doporučováno jej používat například u přehrávačů hudby/videa, či ostatních programů, kde je důležité přesně zobrazovat hotovou práci.

[2] Mapper Dashboard je založen na kostře, kde se pouze mění obsah jednotlivých částí a design je z velké části založen na nativním JavaFX stylu Caspian.

Poslední oblastí, kde by se měla dodržovat konzistence, je konzistence GUI mezi jednotlivými verzemi daného programu. Zde vniká často dilema a vývojář je nucen si položit otázky typu: *„Je možné dále vyvíjet program vytyčeným směrem bez změny GUI?“*, *„Je lepší provést všechny změny v GUI najednou, nebo je postupně přidávat?“* a *„Má smysl zachovat i staré rozhraní pro konzervativní uživatele?“* Odpovědi na tyto otázky nejsou jednoduché a je potřeba si je důkladně promyslet. Přechod na novou verzi s novým GUI bývá totiž pro uživatele (obzvlášť pro ty konzervativnější nebo starší) bolestivý. V posledních letech lze nalézt hned několik příkladů programů, kdy došlo k razantní změně GUI. Mezi změny, které se dotkly veliké masy lidí, můžeme zařadit přechod z Gnome 2 na Gnome 3 (Shell), změna klasického menu v Microsoft Office 2007 na nabídku Ribbon nebo přidání Modern UI (Metro) do Windows 8.

2.4.5 Jednoduchost

V této kapitole nám pomůže výrok Alberta Einsteina: ***„Everything should be made as simple as possible, but not simpler.“***, který se dá pěkně použít i v oblasti tvorby programů. Ve zkratce je tak řečeno, že vlastnosti programu by měly být redukovány do té míry, kdy ještě naše cílová skupina uživatelů bude moci pohodlně a efektivně vykonávat svoji práci. Jakákoli další redukce, či redundance by narušila tuto definici.

Jedná se tedy o to vybrat správnou množinu funkcí, bez kterých se uživatelé neobejdou, správně je reprezentovat a dát na správné místo. A právě toto může být problematické. Je třeba s výběrem těchto funkcí začít již v době analýzy požadavků na program a vést o tom diskuse s uživateli/zákazníkem, protože především ti by měli vědět, co je náplní jejich práce a bez čeho se neobejdou. Pokud se začnou objevovat požadavky typu: *„Já chci věc A dělat způsobem X.“* a *„Já chci stejnou věc A dělat způsobem Y.“*, tak je potřeba zpozornět a rozhodnout, zdali je opravdu nezbytné, aby se ten stejný úkon měl dělat dvěma různými způsoby. Pokud dospějeme k závěru, že není nutné podporovat oba způsoby, vybereme pouze ten, který se nám zdá nejvhodnější.

Další postupy a doporučení, jak dělat programy jednoduché a přitom použitelné jsou k nalezení například ve článku *„Powerful and Simple“* [13].

2.4.6 Klávesové zkratky

Pokud naše aplikace poběží na zařízení, ke kterému je (nebo může být) připojena klávesnice, tak by také měla podporovat klávesové zkratky. Je to z toho důvodu, že zmáčknutí kláves/y je rychlejší než najetí myší na příslušný grafický prvek a kliknutí. Máme-li navíc program, který má ve větší míře přijímat nějaké vstupy od uživatele (okno s formulářem, textový editor, IM klient atp.), může nepodpora/nepoužití klávesových zkratek značně degradovat efektivitu práce uživatele. V některých případech mohou být uživatelé dokonce zmateni, pokud program nereaguje na obecně známé klávesové zkratky.

Bývá zvykem, že při tvorbě klávesových zkratek pro náš program dodržujeme standardy, které jsou vydány pro danou platformu. Uživatelé většinou znají zkratky na platformě, kde pracují a kromě toho, že se nebudou muset učit nové věci, tak se také budou cítit pohodlněji. Dobré také je, pokud aplikačně specifické zkratky nemusí uživatelé hledat v manuálu, ale zobrazí se jim v nápovědě po najetí myší nad daný grafický prvek (Mapper Dashboard k tomuto účelu používá komponentu `Tooltip`. Popis této komponenty je k nazelení v manuálu k JavaFX 2.2 [1]).

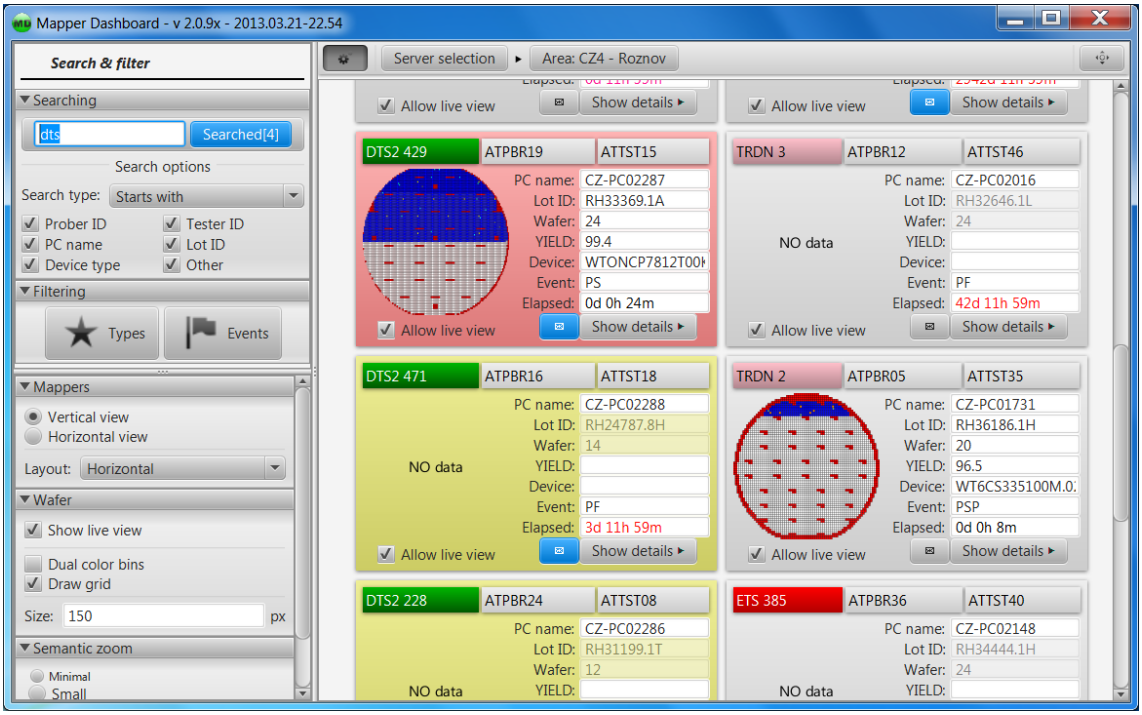
2.4.7 Barvy

Barvy hrají důležitou roli při sdělování informací uživatelům. Pokud je správně použijeme, můžeme uživatelům sdělovat informace bez toho, abychom použili text nebo zvuk. V opačném případě se může stát naše uživatelské rozhraní nepoužitelné.

Použití barev

- **Zvýraznění** - barvu můžeme použít, pokud chceme zvýraznit určitou část v jinak monotóně vypadajícím prostředí a pomoci tak uživateli k rychlejší orientaci a získání důležitých informací. Například pokud ukazujeme uživateli okno s nějakou zprávou/upozorněním, tak použitím odlišné barvy pro určitá slova uživatele upozorníme určité části textu.
- **Stav** - stav určitých objektů může být indikován právě barvou. Například pokud v prohlížeči klikneme na nějaký odkaz, tak ten poté změní barvu a odliší se tím od ostatních odkazů, které vedou na dosud nenavštívené stránky.
- **Význam** - z psychologie je známo, že různé barvy mají pro člověka různý význam. Červenou barvu proto můžeme užít k indikaci chybového stavu a zelenou pro změnu k tomu, abychom uživateli sdělili, že je vše v pořádku.
- **Odlišení** - barvou můžeme také definovat vztah mezi objekty. Uživatelé totiž předpokládají, že objekty z podobné skupiny budou mít také podobné (stejné) barvy. Zároveň použitím rozdílných barev můžeme odlišit například obsah od panelu nástrojů.

Při navrhování barevného schématu pro naši aplikaci si musíme dát pozor na některé věci. Například význam barev není všude na světě stejný (v Japonsku se v minulosti považovala černá barva za svatební a bílá se zase používala při pohřbech) [13, Color]. Také se může stát, že náš program bude používat uživatel s nějakou zrakovou vadou (v populaci má přibližně 8% mužů problém s vnímáním zelené a červené [14]). A v neposlední řadě je nutno počítat s tím, že monitory mají různou kvalitu a zobrazovaní barev (na jednom monitoru může být žlutý text na bílé pozadí čitelný, ale na dalších deseti ne).



Obrázek 2.4: Obrázek Mapper Dashboardu, kde jsou jednotlivé výsledky hledání barevně zvýrazněny. Červené zvýraznění značí fokus.

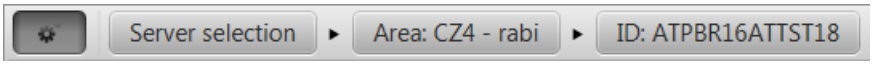
2.4.8 Navigace v programu

Pokud pracujeme na programu, který by se dal označit jako komplexní, případně „prohlížečového typu“ (*browser-based*), je dobré zobrazit uživateli navigaci. Tato navigace by měla být vždy viditelná, snadno dostupná a měla by odpovídat způsobu, jakým se v programu uživatel pohybuje. Tři základní koncepty, které se používají, jsou navigace pomocí stromu, tabů a průvodců.

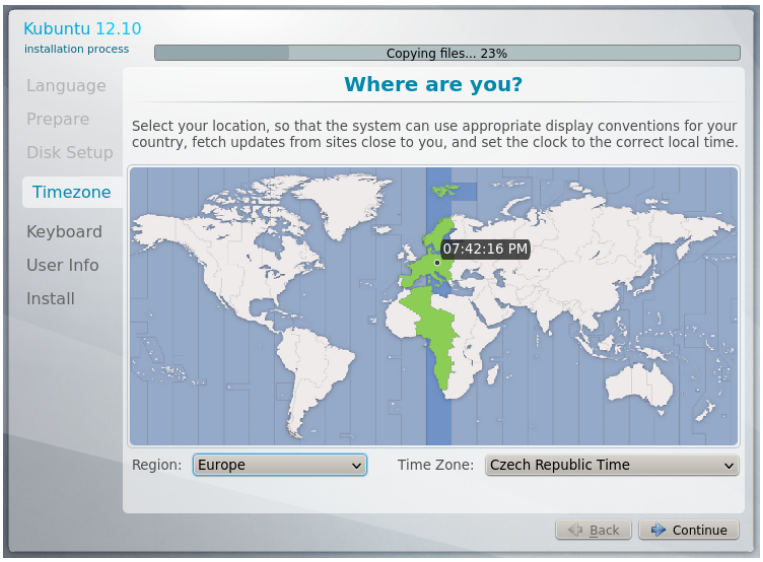
Navigaci pomocí stromu můžeme použít v případě, kdy se uživatel pohybuje po hypotetické cestě. Uživateli ukazujeme cestu, kterou absolvoval, a dáváme mu možnost se vrátit v této cestě nazpět. Tento princip navigace využívají například souborové manažery jako Natilus či Windows Explorer.

Taby (tabbed document interface) se používají v případě, kdy je potřeba mít v jednom okně aplikace otevřeno více dokumentů současně a rychle mezi nimi přepínat. Tento způsob navigace je k nalezení především v textových editorech a webových prohlížečích.

Průvodce (anglicky Wizard/Setup Assistant) je typ GUI, kde je uživatel navigován určitým komplexním procesem. Jedná se v zásadě o sadu dialogových oken/formulářů, které definují jednotlivé kroky/fáze procesu. Pokud v průběhu procesu nedochází k nevratným změnám, tak by měl mít uživatel možnost libovolně přecházet mezi jednotlivými fázemi a upravovat své předešlé volby. Dobrým příkladem tohoto druhu navigace je instalační průvodce v Kubuntu 12.10 (obrázek 2.6).



Obrázek 2.5: Obrázek s navigací z Mapper Dashboardu. Jednotlivé navštívené kontexty jsou řazeny za sebou a je možno se k nim skrze tuto navigaci vrátit.



Obrázek 2.6: Obrázek instalačního průvodce v Kubuntu 12.10.

Kapitola 3

Architektura systému a návrh GUI

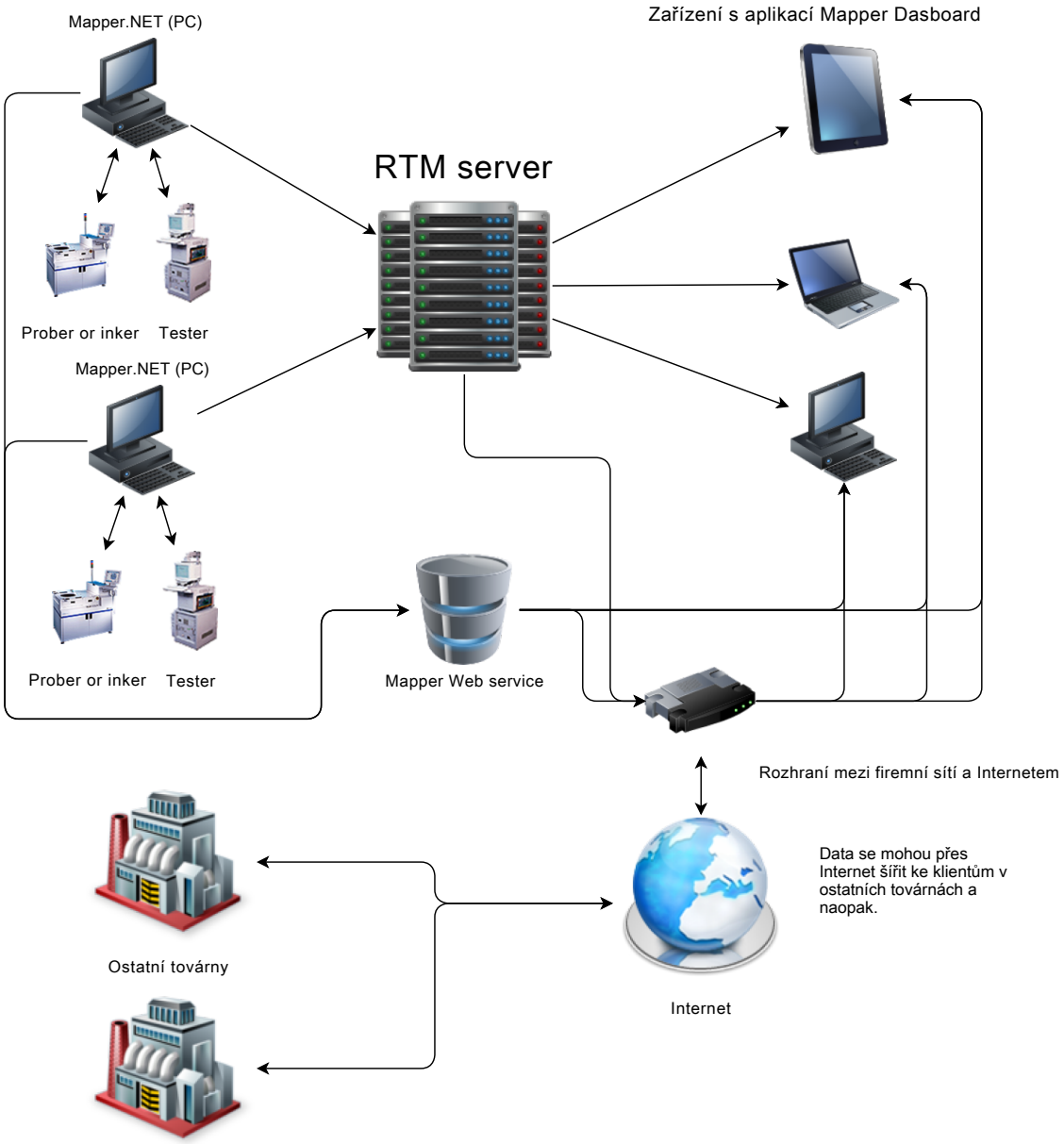
3.1 Architektura systému aplikací

K tomu, aby bylo možno zobrazovat výsledky měření čipů v reálném čase, tak je potřeba dostatečně rychle a efektivně přenášet data z jejich zdroje (aplikace Mapper.NET), až ke klientům (aplikace Mapper Dashboard). Nelze ovšem vytvořit takový systém, kde Mappery budou posílat data přímo klientům. Je to z toho důvodu, že Mapper.NET má za úkol jiné věci a obsluha více klientů by jej mohla nadměrně zatížit.

Proto bylo potřeba přenést zátěž z Mapper.NET na server (RTM server). Ten dočasně uchovává informace o měření jednoho waferu pro každý připojený Mapper.NET a zároveň se stará o to, aby klienti dostávali všechny relevantní informace o měření a to nezávisle na tom, kdy se k RTM serveru připojili.

V průběhu vývoje aplikace Mapper Dashboard se ovšem objevil požadavek na to, aby bylo možno zobrazit základní informace o posledním proběhnutém měření, události a jejím čase. Jako zdroj těchto informací pro Mapper Dashboard nemohl sloužit přímo RTM server, jelikož ten pracoval pouze s připojenými (online) Mappery a navíc si neukládal žádné persistentní informace. Pro získání těchto a dalších informací (jako například poznámky k Mapperům, jejich rozložení v továrně atd.) bylo použito Mapper Web service.

Takto navržený systém aplikací zapadal do systému, který již existoval v Rožnově pod Radhoštěm a byl jednoduše přenositelný do dalších oblastí, kde se používá Mapper.NET verze 3.5 a novější.



Obrázek 3.1: Diagram systému aplikací.

3.2 Real-time Mapping server - podrobnější informace

Základní informace o tom, jaký je účel serveru a jaká technologie byla použita pro jeho tvorbu, byly již zmíněny v kapitole 2.1.1. V této kapitole bude RTM server rozebrán detailněji.

Při návrhu tohoto serveru byly kladeny požadavky především na rychlost a škálovatelnost. Jelikož se ovšem jedná o poměrně specifický server, bylo potřeba v průběhu vývoje vytvořit a implementovat několik rozdílných architektur, porovnat výsledky z reálného provozu a poté zvolit takovou, která bude nejvíce vyhovující.

Plně paralelní architektura

První vytvořená a implementovaná architektura serveru byla plně paralelní. Tedy každý připojený zdroj dat a klient měl přidělené svoje vlákno, které jej obsluhovalo. Navíc tím, že byly použity speciální kolekce pro předávání dat mezi vlákny, bylo přeposílání dat velice rychlé. V zásadě tedy vlákna obsluhující Mappery předávala přijatá data vláknům, která se starala o posílání dat aplikaci Mapper Dashboard. Tato architektura ovšem měla problém s tím, že Mappery sice mohly být připojeny k RTM serveru, ale nemuselo se na nich zrovna pracovat (tedy příslušná vlákna nic nedělala a pouze zabírala RAM paměť na serveru). Pokud by se tento druh serveru nasadil v Asii, kde může být připojeno i 200 Mapperů, tak by docházelo ke zbytečnému plýtvání paměti serveru.

Jednovláknová architektura

Další architektura RTM serveru byla jednovláknová. Implementačně se sice jednalo o 2 stálá a 1 dočasné vlákno, ale prakticky obsluhu všech Mapperů i klientů zajišťovalo jediné vlákno. Tato verze měla výhodu v tom, že na serveru zabírala malé množství paměti (3 vlákna + kolekce s daty a meta-daty) a dokázala posílat data ve shlucích (jeden paket mohl obsahovat více dat, než u plně paralelní verze byla a tedy menší režie s posíláním dat přes síť). I když tato verze dosahovala při reálném nasazení v Rožnově pod Radhoštěm (současně připojených Mapperů bylo do 20 a klientů do 4) dobrých výsledků a data chodila klientům rychle, tak nebylo možno takto navržený RTM server jakkoli škálovat a nebyl využit potenciál více jádrových procesorů.

Hybridní architektura

Po analýze výsledků a vlastností dosažených u předchozích dvou architektur, bylo rozhodnuto vytvořit hybridní návrh. Ten kombinuje výhody předchozích dvou verzí RTM serveru. Stejně jako u plně paralelní verze i tato dokáže plně využít potenciálu moderních více jádrových procesorů, ale přitom nevytváří vlákno pro každé připojení (šetří RAM paměť) a dokáže posílat data ve shlucích. Základní princip této architektury je postaven na tom, že administrátor předá RTM serveru jako argument počet vláken v `thread-poolu` a RTM server dokáže sám lineárně rozdělit zátěž mezi jednotlivá vlákna. Počet vláken by se měl volit podle předpokládaného počtu Mapperů v továrně a volných zdrojů serveru. Pokud tedy počítáme s tím, že na RTM server bude připojených 60 Mapperů a server nemá dostatečné zdroje pro vytvoření 60 vláken, tak jich můžeme vytvořit jen 10 a RTM server si sám rozdělí Mappery mezi jednotlivá vlákna.

3.3 Mapper Dashboard - podrobnější informace

3.3.1 Požadavky na aplikaci

Oficiální zadání a požadavky na tento software byly spíše obecnějšího rázu (*„Lightweight modulární aplikace, která dokáže zobrazovat průběhy měření v reálném čase a bude napsána na platformě JavaFX.“*), což mi dalo volnou ruku při jejím návrhu a následné implementaci. Nicméně i z takto obecného zadání následně vyplynuly další požadavky a cíle, jichž by aplikace měla dosáhnout.

- **Uživatelská přívětivost** - aplikace musí být dobře použitelná a ovladatelná i pro méně zkušené uživatele. GUI by mělo být navrženo jednoduše a přehledně.

- **Spolehlivost** - aplikace nesmí dávat špatné výsledky/špatně zobrazovat, protože by to mohlo způsobit zmatky.
- **Lightweight** - poněkud problematický požadavek, jelikož existuje několik různých interpretací toho, co slovo *„lightweight“* znamená. V kontextu aplikace Mapper Dashboard byl tento požadavek interpretován tak, že výsledek práce musí být rychlý, dobře sloužit svému účelu (tedy především zobrazovat wafery) a nemít přehnané nároky na hardware.
- **Modularita** - aplikace se musí dát v budoucnu jednoduše rozšiřovat.
- **Aktuálnost** - aplikace by se měla umět sama aktualizovat. Tedy bez jakéhokoli zásahu, či pozornosti ze strany uživatele.

3.3.2 Základní rozvržení aplikace

Základní princip fungování aplikace Mapper Dashboard je založený na kostře a „kontextech“, které tuto kostru vyplňují (více v kapitole 3.3.3). Kostra je prakticky neměnná po celou dobu běhu programu, ovšem v závislosti na kontextu, ve kterém se uživatel právě nachází, se mění obsah některých částí kostry. Demonstrativní příklad s kontextem náhledu na mapy továrny v Rožnově pod Radhoštěm je ukázán na obrázku 3.2. Pro účely popisu byly jednotlivé části barevně zvýrazněny a odlišeny.

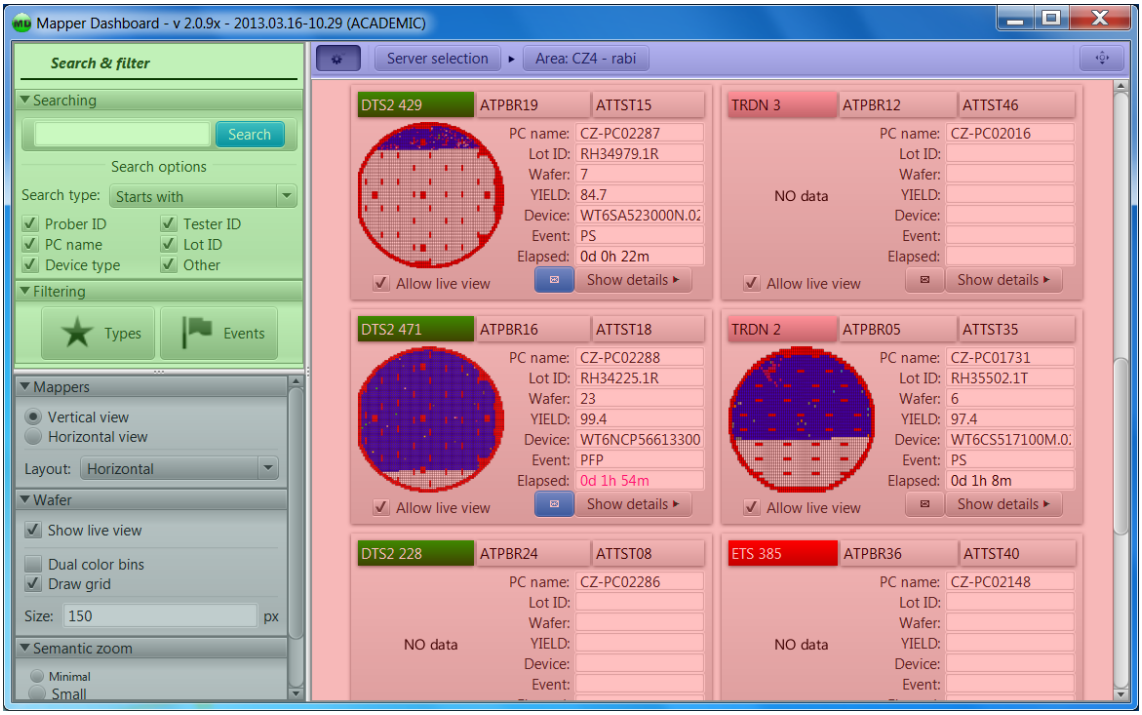
Horní modrá část obsahuje tlačítko (na levé straně), kterým se dá schovat/zobrazit boční panel, dále pak seznam kontextů navštívených uživatelem (první si uživatel vybere server (továrnu), ke kterému se chce připojit a poté se mu zobrazí mapa Mapperů), na pravé straně je panel nástrojů pro daný kontext (v tomto kontextu prázdný) a tlačítko, které přepne aplikaci do režimu celé obrazovky (toto tlačítko zůstává ve všech kontextech). Mezi jednotlivými kontexty se dá přecházet tak, že uživatel klikne na tlačítko s názvem daného kontextu.

Největší část plochy je vyznačena růžovo-červenou barvou. Jedná se o místo určené pro obsah (na demonstračním obrázku je obsahem rozložení továrny v Rožnově p.R.). Pokud je obsah vkládaný do této části větší, než plocha jemu přidělená na monitoru, tak se aktivují posuvníky, pomocí kterých se uživatel může dostat i ke zbytku obsahu.

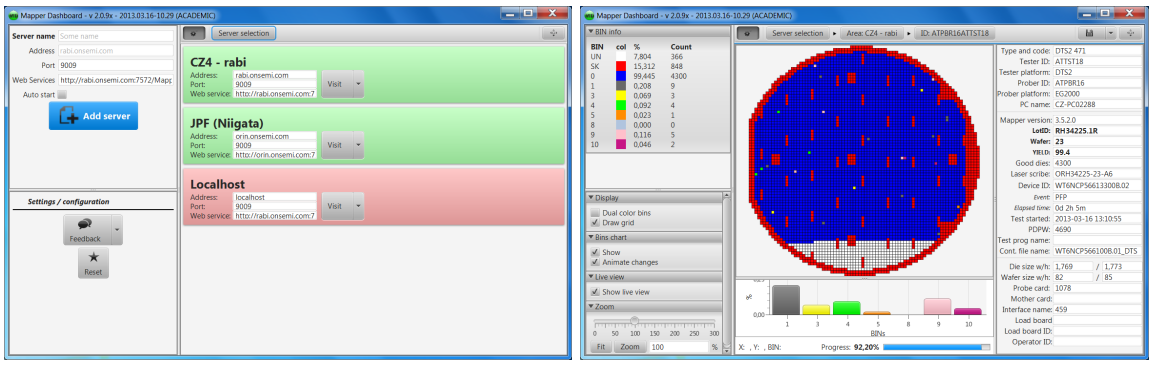
V levé části je boční panel, který je v základu rozdělen na dvě části. Spodní část (zvýrazněná tmavě zeleně) slouží pro různá nastavení, která se vztahují k růžové části. Horní světle-zelená poté může obsahovat další věci specifické pro daný kontext. V tomto případě nástroje pro vyhledávání v Mapperech a jejich filtrování (v kontextu detailního náhledu na vybraný Mapper to je pro změnu tabulka s počtem BINů a jejich barvami).

Takto navržená kostra, spolu s kontexty zajišťuje modularitu a snadnou rozšiřitelnost programu. Vývojář prakticky jen vytvoří třídy, které implementují určité rozhraní a Mapper Dashboard se už sám postará o to zbytek.

V současné době (březen 2013) jsou vytvořeny 3 kontexty. V prvním si uživatel může vybrat, ke kterému serveru se připojí. Ve druhém vidí mapu Mapperů v jedné továrně. A ve třetím kontextu jsou zobrazeny detailní informace o jednom Mapperu, který si uživatel vybral z mapy Mapperů (viz obrázek 3.3).



Obrázek 3.2: Základní rozvržení Mapper Dashboard.



Obrázek 3.3: Ukázka kontextů výběru serveru a detailního náhledu na Mapper.

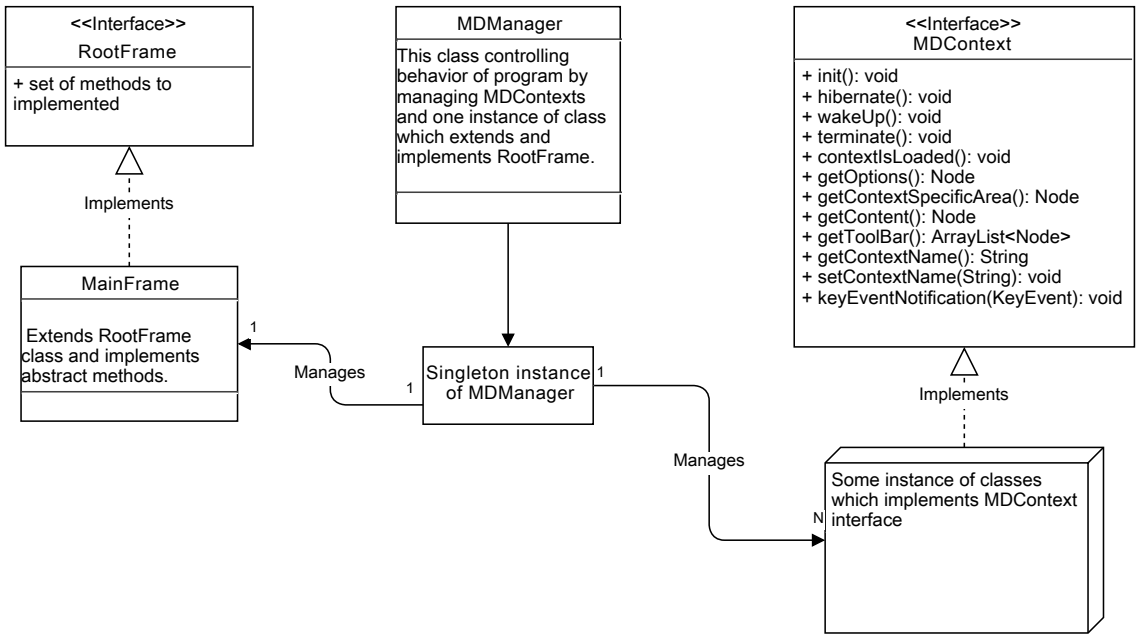
3.3.3 Kontexty

Systém kontextů byl vytvořen kvůli požadavku na modularitu aplikace Mapper Dashboard a její snadnou rozšiřitelnost v budoucnu.

Základ vnitřní architektury funguje tak, jak je znázorněno na obrázku 3.4. Existuje rozhraní `MDContext`, jenž definuje metody, které musí příslušné třídy implementovat, chtějí-li zobrazit svůj obsah v kostře aplikace. Kostra aplikace je pro změnu povinna implementovat rozhraní `RootFrame`.

Tím, že byla použita rozhraní, tak programátoři kontextů, případně koster aplikace dostali velikou svobodu v implementaci. Pokud se tedy někdo rozhodne, že by bylo dobré přesunout současný boční panel nahoru, seznam kontextů pro změnu na levou stranu a

panel nástrojů na stranu pravou, tak není problém. Stačí implementovat příslušné rozhraní `RootFrame` a GUI kostry programu (případně poupravit kostru stávající).



Obrázek 3.4: Diagram použití kontextů.

Výhoda této architektury je také v tom, že kostra a kontexty spolu přímo nekomunikují. Veškerou komunikaci, respektive řízení chování, zajišťuje instance třídy `MDManager` (vnitřně implementovaná vzorem *singleton*). Tato třída se stará o změny kontextů, jejich hibernaci, probouzení, smazání atp. Také je v ní obsažena reference na objekt scény celého programu. Je tak možno zachytávat například stisky kláves (klávesové zkratky) a informovat o této skutečnosti aktivní kontext.

3.3.4 Odstranění drop-down menu

Odstranění drop-down menu se z počátku jevilo jako poměrně radikální a riskantní krok. Důvod byl ten, že většina aplikací, které uživatelé používají v ON Semiconductor toto menu má a uživatelé obecně neradi mění své zvyky.

U aplikace Mapper Dashboard by ovšem drop-down menu znamenalo problém (schizma kontextovosti a globálnosti). Důvod je ten, že samotné základy jsou postaveny na práci v jednotlivých „kontextech“. Každý kontext má svůj účel a své vlastní nastavení/volby/nástroje. Uživatelé tedy v daném kontextu naleznou vše, co by mohli pro práci s ním potřebovat. Pokud nějaké nastavení/funkcionalitu nenaleznou v bočním panelu nebo v panelu nástrojů, tak příslušná věc neexistuje a nemá smysl ji dlouho a složitě hledat v menu, kde je nastavení i pro ostatní části programu.

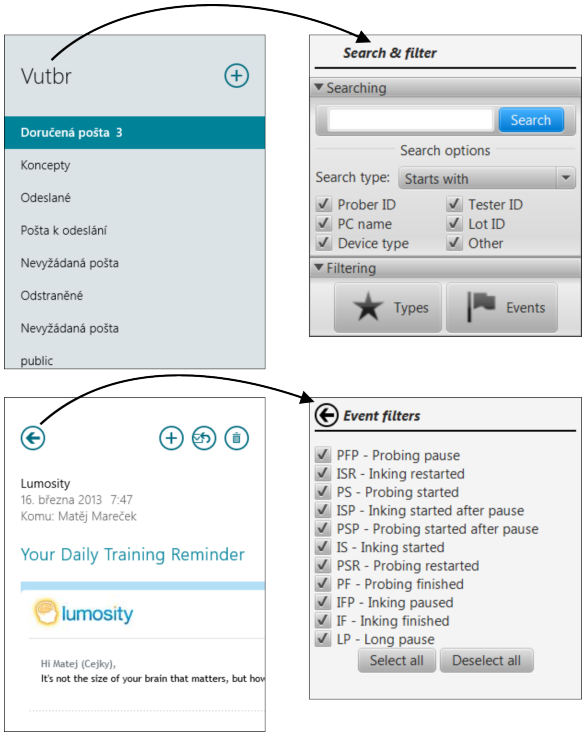
Tím, že uživatel pracuje vždy v jednom kontextu, tak zároveň dochází k tomu, že se může soustředit pouze na jednu činnost a není rozptylován něčím jiným. Zároveň tak odpadá nutnost tvorby globálního nastavení, které by se muselo s přidáním, či změnou každého kontexty upravovat a díky svému rozsahu by se stalo pro uživatele nepřehledné. Mapper Dashboard by tak také přišel o možnost udělat na uživatele dojem jednoduché, snadno

používané aplikace s jednotným *workflow*.

3.3.5 Metro-style widget

Metro-style widget (zkráceně metro widget nebo jen widget) je odpovědí na otázku: *„Jak zachovat uživatelské rozhraní jednoduché a zároveň poskytnout uživateli pokročilé funkce a nastavení?“* Je to vlastně grafický objekt, do kterého můžeme vložit více úrovní (nazvané metro widget level/metro level) s různým obsahem. Jednotlivé úrovně se přitom mohou zanořovat do sebe a tvořit tak stromovou strukturu. Tímto způsobem je možno recyklovat určitou plochu (oblast na monitoru) a ušetřit tak místo pro jiné věci, které uživatel potřebuje vidět.

Tato grafická komponenta je částečně inspirována Metro (Modern UI) prostředím ve Windows 8. Odtud také pochází její název. Ovšem na rozdíl od Metra, které Microsoft koncipoval jako designově čisté a vzdušné prostředí určené především pro dotykové ovládaní, metro-style widget (přestože má podobnou strukturu) je koncipován jako grafický prvek pro desktopové aplikace, který má za úkol šetřit místo a logicky třídit obsah.

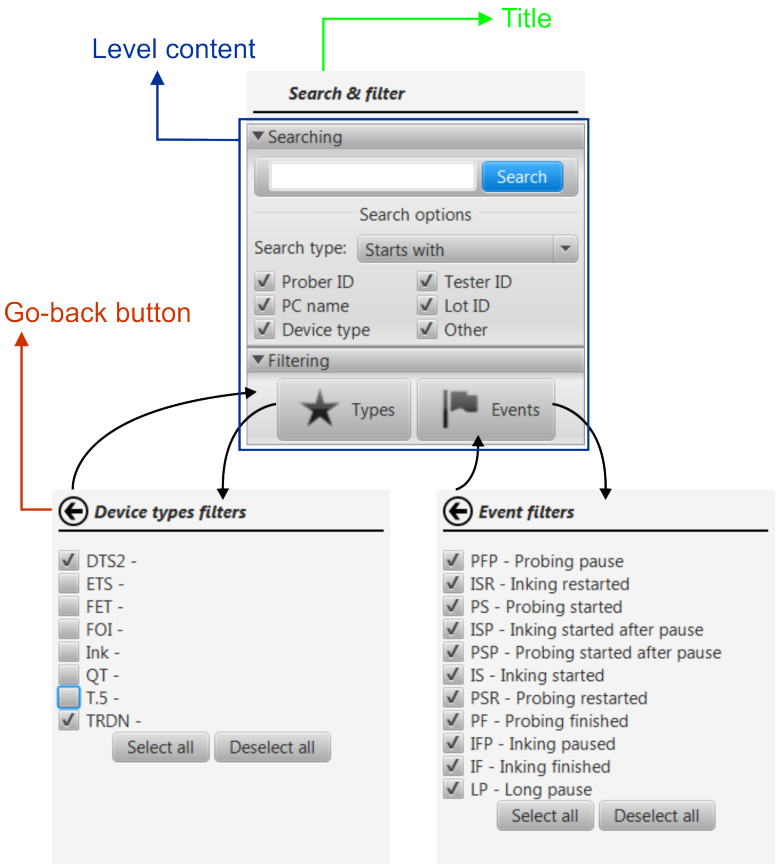


Obrázek 3.5: Původ metro style widgetu.

Ukázka metro-style widgetu je na obrázku 3.2, kde je zvýrazněný světle zelenou barvou. Jeho fungování je poté ilustrováno na obrázku 3.6. V tomto případě (kontext mapy Mapperů) bylo potřeba na relativně malou plochu vměstnat jak vyhledávání, tak různé druhy filtrů. Vyhledávání bylo vyhodnoceno jako funkce, kterou budou uživatelé používat častěji než filtrování. Navíc se její grafická reprezentace (i s různými vyhledávacími preferencemi) vlezla na vymezenou plochu. Zbylé místo bylo využito k umístění tlačítek, jejichž účelem bylo nahrát do příslušného metro widgetu nový metro level, který obsahoval grafickou reprezentaci filtrů.

Jakmile dojde k nahrání nové úrovně do widgetu, tak se změní nadpis (text pro nadpis se získá z nahrávané úrovně a na obrázku 3.6 je označený jako „*Title*“) a přibude šipka směřující na levou stranu, která indikuje, že se pomoci ní může uživatel vynořit o úroveň výše. Pokud je uživatel v kořeni pomyslného stromu, šipka zmizí a zůstává jenom nadpis současné úrovně.

Podrobnosti ohledně implementace metro-style widgetu jsou k nalezení v kapitole 4.3.1.



Obrázek 3.6: Metro-style widget.

3.3.6 Priorita vykreslování GUI

K tomu, aby se aplikace uživateli příjemně používala, je potřeba zajistit dostatečně rychlé reakce a plynulost GUI. Teoretické základy tohoto tématu (včetně studií a doporučení, které se zaměřovaly na dobu odezvy) byly popsány v kapitole 2.4.3.

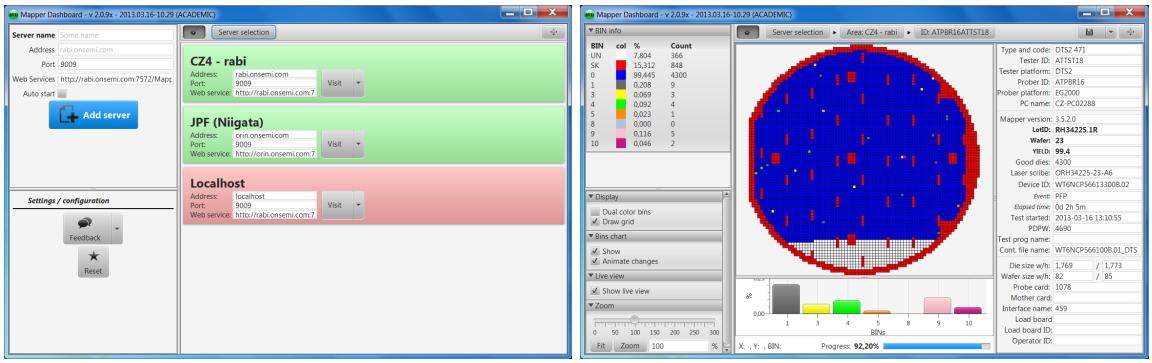
Vykreslování průběhu měření čipů prováděné na waferech v reálném čase bylo i s provedenými optimalizacemi výpočetně náročné, a pokud uživatel například měnil velikost bočního panelu, nebo posouval pohled v mapě Mapperů (pokud byla tato mapa dostatečně velká a nevešla se do místa vyhrazeného na obsah), mohl pociťovat pomalejší reakce GUI a také určitou nekontinuitu při pohybech (drobná zasekávání, která ovšem byla velice nepříjemná).

Tyto problémy s plynulostí a odezvou uživatelského rozhraní byly způsobeny tím, že vykreslovací vlákno v JavaFX vykreslovalo především průběh měření. Pokud chtěl uživatel například změnit velikost okna, tak na jeho požadavek vykreslovací vlákno reagovalo až v

okamžiku, kdy vykreslilo všechny čipy, které mělo naplánováno zobrazit (to způsobovalo počáteční pomalejší odezvu GUI). Druhý problém s drobnými záseky byl způsoben tím, že i když uživatel pracoval s GUI (viz demonstrativní příklad se změnou velikosti okna), tak se GUI vlákno zároveň s tím snažilo vykreslovat data přicházející z RTM serveru (a právě tato operace mohla při větším množství dat trvat neúměrně dlouho a uživatel byl vyrušen uprostřed svojí činnosti).

Řešení těchto problémů se podařilo nalézt v podobě vlákna, které rozhoduje o tom, jaké činnosti (většinou se jedná o změnu nějaké GUI komponenty nebo vykreslení změřených čipů) se naplánují pro budoucí vykonání ve vykreslovacím vlákně aplikace (také se tím řeší další problém popsaný v kapitole 4.3.2). Toto vlákno zároveň implementuje rozhraní, které mu dovoluje stát se posluchačem událostí (včetně priority těchto událostí), na které reaguje tak, že pozastaví svojí plánovací činnost. Tím pádem přestane zatěžovat vykreslovací vlákno (to se poté může nerušeně věnovat interakci s uživatelem).

Jakmile nejsou detekovány žádné požadavky na interakci s uživatelem (čeká se standardně 1 sekundu), tak plánovací vlákno obnoví svoji činnost[1].



Obrázek 3.7: Ukázka kontextů výběru serveru a detailního náhledu na Mapper.

3.4 Zpětná vazba od uživatelů a hlášení chyb

V průběhu vývoje aplikace Mapper Dashboard docházelo s uživateli poměrně často k výměně emailové korespondence ohledně různých chyb, návrhů na vylepšení/novou funkcionalitu atp. Problém s tímto způsobem komunikace nastával v případě, kdy uživatelé nahlašovali nějakou chybu, případně nestandardní/neočekávané chování programu. V takových situacích bylo důležité zjistit nejen způsob, jakým bylo s aplikací manipulováno, ale to co se dělo uvnitř. Tedy projít logovací soubory.

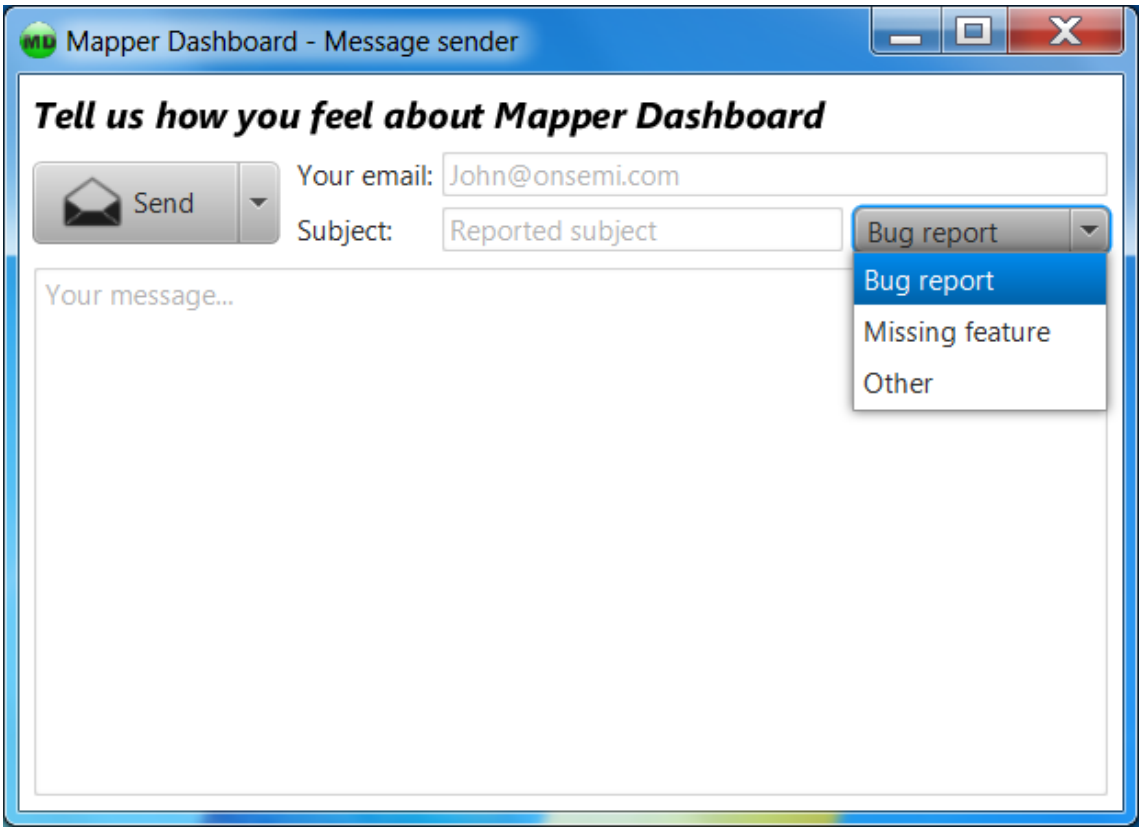
Vzhledem k tomu, že není možné požadovat po uživatelích znalosti ohledně logování a konfigurace počítače, na kterém právě pracují (typ a verze operačního systému, virtuálního stroje Javy atp.), tak samotné posílání logovacích souborů jako emailové přílohy není také příliš pohodlné. Navíc ve většině případů nebývá vhodné ukázat uživatelům, kde naleznou složku s interními soubory aplikace.

K tomu, aby byli uživatelé motivování k nahlašování různých chyb a ke zpětné vazbě obecně, bylo zapotřebí tento proces co nejvíce zjednodušit a zpříjemnit. Cílem bylo, aby

[1] Tento systém je řízen skrze synchronizované rozhraní instance třídy `MDManager` a kromě toho, že jednotlivé části programu mohou reagovat na požadavky omezením své činnosti, tak také mohou zaslat posluchačům požadavek o určité prioritě.

uživatelé neměli pocit, že ztratili 10-15 minut svého času popisováním něčeho, co se jim právě přihodilo, ale aby naopak tento úkol zvládli jednoduše během několika minut a měli dobrý pocit z toho, že pomohli s vylepšením programu.

Proto vznikl `Mapper Dashboard - Message sender` (obrázek 3.8). Jednoduchý vestavěný emailový klient (součást aplikace Mapper Dashboard), který umožňuje uživatelům pohodlně zasílat hlášení o chybách, chybějící funkcionalitě, případě jiný typ zprávy. Zároveň s tím se odešle obsah logovacího souboru, IP adresa počítače a další informace, které mohou pomoci support teamu při řešení a nápravě případných problémů.



Obrázek 3.8: Okno pro získání zpětné od uživatelů.

3.5 Protokol pro komunikaci a přenos dat

Komunikace mezi aplikacemi Mapper.NET, Mapper Dashboard a Real-time Mapping serverem je prováděna skrz TCP/IP spojení. K přenosu informací se využívá textový protokol s hierarchickou strukturou (je to z důvodu jednoduché a efektivní syntaktické analýzy). Každá zpráva je standardně ukončena znakem "$\backslash n$".

Hierarchie protokolu je tvořena v těle zprávy oddělovači. Oddělovacím znakem je: `">"`. Na konci této hierarchie mohou/nemusí být další argumenty (záleží na sémantice zprávy). Například zpráva `"MI>LotEnded>\n"` nám říká, že skončilo měření lotu a zpráva `"MI>LotId>" + lotId + "\n"` zase říká, jaký lot se právě měří.

Kapitola 4

Implementace Real-time Mappingu a vyhodnocení výsledků

4.1 Mapper.NET

4.1.1 Implementace zasílání dat na RTM server

O zasílání dat na RTM server se v Mapper.NET stará samostatné vlákno. To je vytvořeno při spuštění aplikace a informace o tom, na jaký server a jaký port se má připojit bere z konfigurace. Vnitřně poté celý proces funguje na komunikaci tří tříd.

První třídou je `RealTimeMapManager`. Ta byla zabudována přímo do existujících struktur Mapper.NETu a funguje na bázi reakcí na událostí (`Events`). Pokud v Mapperu dojde k nějaké události (začíná nové měření, je změřen/zakapán čip, přestalo se měřit atd.), tak je tento manažer informován a je v jeho režii, jak s příslušnou informací naloží (při volání příslušné metody se předává reference na objekt odesílatele a události). Dojde-li k rozhodnutí, že by se na danou událost mělo reagovat, tak manažer zavolá příslušnou metodu instance třídy `RealTimeMapWriter`.

Třída `RealTimeMapWriter` je v zásadě konvertor, který převádí volání svých metod a argumenty událostí do formy textového protokolu/dat, kterým se komunikuje s RTM serverem a Mapper Dashboard klientem. Zároveň s vytvořením instance této třídy dochází k vytvoření nového objektu třídy `RTMServerConnector`, kterému se předávají data k odeslání.

`RTMServerConnector` je posledním článkem v tomto řetězci volání. V jeho konstruktoru (jemuž se předávají argumenty: název serveru a číslo portu, ke kterému se připojit) dojde k vytvoření a nastartování vlákna, které se stará o zasílání dat na RTM server. K bezpečnému zapisování/čtení dat do/z instance této třídy slouží speciální kolekce zvládající přístup z více vláken zároveň. Aby vlákno nečekalo v případě nečinnosti Mapperu na výběr dat z kolekce, tak je použit několikasekundový časový limit, po jehož vypršení je zaslána na RTM server *„keepalive“* zpráva, ta říká, že Mapper a připojení jsou v pořádku, ovšem nic se neměří. Pokud RTM server nedostane tuto, či jinou zprávu, tak síťové spojení s Mapperem po určité době ukončí.

Dojde-li k problému na síti a Mapper je odpojen od RTM serveru, tak dochází k pravidelným pokusům o znovunavázání spojení. Po opětovném navázání spojení Mapper znovu odešle data již odeslaná[1] a také data, která ještě odeslaná nebyla (ta jsou v kolekci sloužící k předávání dat mezi vlákny), ale odpovídají aktuálnímu stavu Mapperu.

[1] Odeslaná data se neukládají všechna. Pokud dojde ke změně waferu, tak odeslaná a uložená data již nekorespondují se současným stavu Mapperu a jsou vymazána.

4.2 RTM server

4.2.1 Hlavní vlákno

Hlavní vlákno RTM serveru má na starost pouze 2 věci.

Zaprvé hned po spuštění programu zpracuje argumenty a výsledky této operace uloží do instance třídy `RTMServerSettings` (návrhový vzor *singleton*). Dále nastaví třídě `Thread` handler nezachycených výjimek (aby nedošlo k vypnutí serveru) a vytvoří `ServerSocket`.

Druhou činností tohoto vlákna je přijímání příchozích spojení. Většinu doby se tedy čeká v metodě `accept()`. Jakmile tato metoda vrátí nový soket s připojením, tak se vytvoří a spustí instance třídy `DecisionThread` (ta je odvozená od třídy `Thread` a argumentem konstruktoru by měl být soket s právě přijatým připojením). Ta si od klienta přečte první řádek dat a rozhodne, jak a zdali vůbec odpoví.

4.2.2 Rozhodovací vlákno

Rozhodovací vlákno provádí rozhodnutí na základě shody prvních několika písmen v řetězci, který se přijme od klienta:

- *„Mapper“* - pro klienta se vytvoří instanci třídy `MapperSession`, která je vložena do vlákna `ManagerThread`.
- *„Client“* - klientem je tady Mapper Dashboard, kterému bude RTM server zasílat data a vytvoří se pro něj instance třídy `ClientSession`. Tato instance je také vložena do vlákna `ManagerThread`.
- *„Status“* - klientovi se odpoví, jaký je současný stav serveru. Pro zatím je jen stav *„online“*, ale v budoucnu se může tato odpověď rozšířit i o stavy *„debugging“*, *„error“* atp.
- *„Info“* - klientovi se pošlou informace o serveru (kdy byl server spuštěn, kolik je k němu připojených zdrojů dat, klientů atd.).
- *Jiné* - klientovi není zaslána žádná odpověď a informace o nerozpoznaném připojení jsou uloženy do logovacího souboru.

Jakmile toto vlákno klientům ne/odpoví, nebo je předá do `ManagerThread`, nastane proces jeho ukončení.

4.2.3 Příjem a přeposílání dat

O příjem a správné přeposílání dat se stará vlákno s názvem `ManagerThread` (vzor *singleton*). Do něj se synchronně vkládají instance tříd `MapperSession` a `ClientSession`, které reprezentují jednotlivá připojení.

Zjednodušeně se dá říci, že práce tohoto vlákna spočívá ve správě jednotlivých sezení (periodicky se kontroluje, jestli nová sezení nebyla vytvořena, či ukončena) a distribuci úkolů mezi další vlákna (k tomuto je použit `ThreadPool` o fixní velikosti). O správnost dat, která mají být zaslána klientům, se starají jednotlivé instance třídy `MapperSessin` (ty si k tomuto účelu budují příslušné metainformace).

Ke zjištění detailnějších informací o této třídě doporučuji prostudovat soubor `ManagerThread.java`.

4.3 Mapper Dashboard

4.3.1 Implementace metro-style widget

Metro-style widget je implementován jako skupina kooperujících tříd, jejichž účel byl popsán v kapitole 3.3.5. V následujícím textu bude popsána implementace dvou tříd: `MetroWidget` a `MetroWidgetLevel`. Více detailnějších informací je možno získat přímo ve zdrojových kódech.

MetroWidget

Tato třída byla vytvořena odvozením od třídy `AnchorPane`[1]. Tím pádem je možno instanci třídy `MetroWidget` vložit do *Scene Graphu* a nechat ji vykreslit.

Interně se tato třída skládá z kolekce, která obsahuje jednotlivé úrovně (`MetroWidgetLevel`) a instancí grafických komponent (text pro nadpis, atp.). Tyto komponenty mají privátní přístup a třída si je spravuje sama. Programátor používající `MetroWidget` je tak odstíněn od detailů a je mu ušetřena práce.

MetroWidgetLevel

Tato třída je základní stavební kámen, který obsahuje 3 složky.

- **Jméno úrovně** - jméno úrovně je řetězec, který poté třída `MetroWidget` zobrazí v napisu (pokud daná úroveň právě aktivní).
- **Grafický obsah** - jakýkoli objekt typu `Node`[1]. Ten reprezentuje obsah, který bude uživateli zobrazen.
- **Reference na MetroWidget** - pomocí této reference se pozná, ve které instanci třídy `MetroWidget` je úroveň vložena (pokud není úroveň nikde vložena, je reference `null`).

4.3.2 Vizualizace dat v reálném čase

Vizualizace dat v reálném čase byla jedna z nejtěžších implementačních věcí v celé této práci. Zaprvé v době, kdy se začínalo s tvorbou Mapper Dashboardu (respektive dělaly se pokusy s JavouFX), tak byla k dispozici pouze stabilní verze JavyFX 2.0. Ta v sobě neobsahovala komponentu `Canvas`, ale uměla pouze vykreslovat objekty v *Scene Graphu*[2] (`Button, Rectangle, AnchorPane` atd.). Problém spočíval v tom, že nelze vytvořit stromovou hierarchii, kde jsou statisíce objektů (počítáme-li s tím, že jeden čip je znázorněný jako jedna instance třídy `Rectangle`) a počítat s tím, že ji vykreslíme (v reálném čase) i na nějakém starším korporátním počítači.

Tyto potíže se podařilo vyřešit až s příchodem JavyFX 2.2, která výše zmíněnou komponentu `Canvas` obsahovala. Bylo tak možné kreslit bez toho, aby se přidávalo nadměrné množství objektů do *Scene Graphu*.

Další problém byl v tom, jak vůbec vykreslit data, která stahuje samostatné vlákno z RTM serveru. JavaFX (stejně jako většina podobných platforem) dokáže vykreslovat grafiku pouze v GUI vlákně aplikace a toto vlákno může být jen jedno. Způsob, jakým se dají provádět určité operace v GUI vlákně je pomocí metody `Platform.runLater(java.lang.Runnable`

[2] Více informací na: http://docs.oracle.com/javafx/2/scenegraph/jfxpub-scenegraph.htm

`runnable)`. Té se předá objekt implementující rozhraní `Runnable` a metoda `Run()` se poté vykoná v GUI vlákně.

Bohužel není možné volat metodu `runLater(...)` příliš často a mnohokrát po sobě. V takovém případě totiž začne JavaFX požadavky na vykreslení „ignorovat“. Oracle nezveřejnil všechny zdrojové kódy JavyFX, takže nebylo možno zjistit, v čem byl přesně problém. Předpoklad je ale takový, že pokud se GUI vláknu nahromadí hodně požadavků do fronty, tak je začne zahazovat. Problém je v tom, že toto nestandardní chování není nikde indikováno, takže se programátor nemá možnost dozvědět, že něco nebylo provedeno.

V průběhu tvorby Mapper Dashboardu bylo vyzkoušeno několik způsobů, jak spolehlivě vykreslovat velké množství čipů v reálném čase, přičemž nejlepších výsledků se dosáhlo s řešením znázorněným na obrázku 4.1. Tento princip je založen na třech vzájemně kooperujících vláknech (vlákno na stahování dat, řízení zátěže GUI vlákna a samotné GUI vlákno).

Vlákno, které stahuje data z RTM serveru, je zároveň ukládá do příslušných objektů (o to, kde jaká data uložit se stará instance třídy `MappersStorage` implementující rozhraní `RTMDataListener`). Tyto objekty jsou implementovány tak, že data pouze uchovávají, ale nesnaží se je vykreslit.

`GUIRefresher` je vlákno, které běží paralelně s ostatními a v zásadě čeká na to, až stahovací vlákno uloží data tam, kam má. Jakmile k tomuto dojde, je `GUIRefresher` vlákno probuzeno a vezme si seznam všech objektů, kde byla přidána, či modifikována nějaká data. Tyto objekty musí povinně implementovat metodu `refreshGUI()`, která se postará o vykreslení nových dat. Kromě toho, že se data vykreslují ve shlucích (redukuje se počet volání metody `runLater(...)` tím, že se najednou vykreslí všechna příchozí data pro daný objekt, což můžou být i tisíce čipů), tak GUI refresher čeká, dokud GUI vlákno neprovede naplánovanou činnost[3].

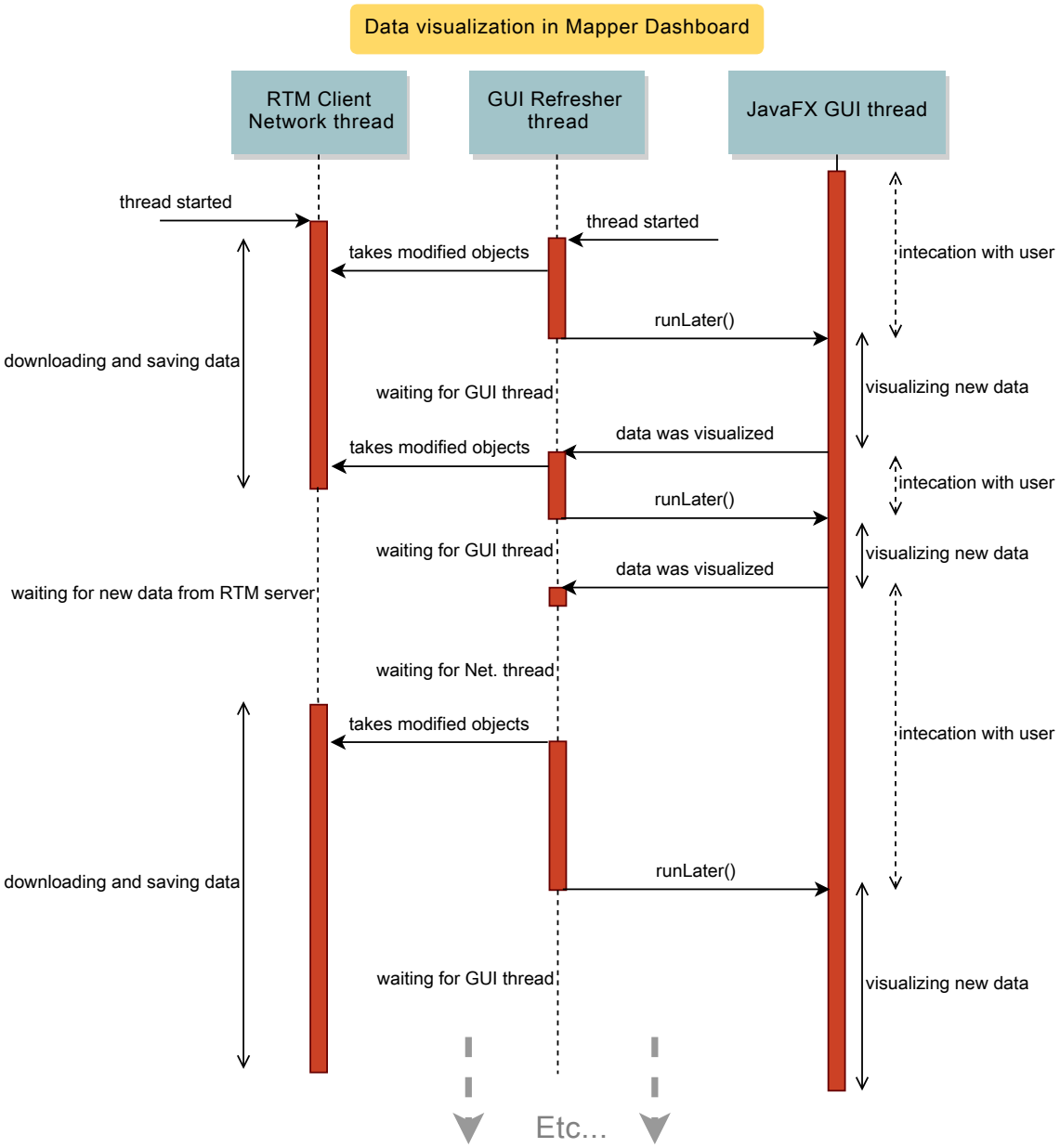
Další výhodou toho přístupu k práci s daty a vykreslováním je, že lze jedním nezávislým vláknem kontrolovat zátěž, která je kladena na vykreslovací vlákno v JavaFX (viz kapitola 3.3.6, která řeší odezvu GUI).

Dosažené výsledky vykreslování

Výše popsaným způsobem se podařily vykreslit desítky až stovky současně měřených waferů (simulováno nebylo více jak 200 Mapperů, jelikož více se v současné době v žádné z továren ON Semiconductor nevyskytuje). Testování probíhalo na stroji Lenovo ThinkPad T500[4] s nainstalovaným operačním systémem Microsoft Windows 7 x64.

[3] Čekání na dokončení naplánované činnosti v GUI vlákně je udělané tak, že objekt implementující rozhraní `Runnable` obsahuje v sobě další objekt, na kterém `GUIRefresher` vlákno zavolá metodu `wait()`. Tedy čeká na notifikaci. Tuto notifikaci poté provede GUI vlákno na konci metody `Run()`.

[4] ATI Radeon 3650, 4GB RAM, Intel Core 2 Duo T9600 (2,8 GHz).



Obrázek 4.1: Spolupráce vláken při stahování a vykreslování dat.

Kapitola 5

Závěr

Cílem této práce bylo především navrhnout a implementovat řešení (softwarový systém nazvaný RTM), pomocí něhož by uživatelé mohli sledovat průběhy měření křemíkových desek v reálném čase.

Vyvinutý softwarový systém se skládá ze čtyř částí. Dvou zdrojů dat, jedné serverové aplikace a jedné klientské aplikace.

Zdrojem dat o probíhajících měřeních v čistých prostorách jsou počítače s nainstalovanou aplikací Mapper.NET. Tato aplikace řídí měření a naměřené výsledky zasílá na RTM server, který si je dočasně ukládá do paměti a v případě potřeby přeposílá klientům. Klientskou aplikací je Mapper Dashboard, jenž si stáhne persistentní data o Mapperech z Mapper Web service. Mapper Dashboard z těchto dat vytvoří virtuální mapu čistých prostor, ve které se vizualizují data přijatá z RTM serveru.

Celý tento systém aplikací byl po několik měsíců testován zaměstnanci firmy ON Semiconductor a na základě jejich zpětné vazby postupně upravován do výsledné podoby. Ta byla poté nasazena do produkčního prostředí v Rožnově pod Radhoštěm a Niigatě (Japonsko), což považuji za veliký úspěch, kterého bylo dosaženo.

V rámci této práce se také podařilo vymyslet způsob, jakým lze na platformě JavaFX 2.2 vykreslovat větší množství grafiky. Toho bylo dosaženo pomocí vzájemné kooperace několika vláken a lepším rozložením zátěže kladené na vykreslovací vlákno v Javě. Mapper Dashboard tak dokáže vykreslovat desítky až stovky současně měřených waferů a to i na strojích, které lze v dnešní době považovat za podprůměrně výkonné.

Také se podařilo navrhnout a implementovat hybridní architekturu serveru pracujícího s dlouhodobě připojenými klienty i zdroji dat. Výhodou této architektury je škálovatelnost, díky níž můžeme pouhou změnou parametrů vytvořit jak nenáročný server zabírající relativně málo paměti, tak server s desítkami vláken, který dokáže plně využít moderních vícejádrových procesorů.

Práce byla otestovaná jak v uměle simulovaných, tak reálných podmínkách a uživatelské rozhraní aplikace Mapper Dashboard bylo uzpůsobeno potřebám uživatelů. Výsledná implementace byla schválena pro použití na produkčním prostředí. Cíl této práce tedy považuji za úspěšně splněný.

U projektu RTM se v budoucnu počítá s přidáním možnosti ovládat vzdáleně Mapper.NET skrze Mapper Dashboard. Tato funkcionalita by měla usnadnit práci jak vývojářům Mapperu, tak test inženýrům.

Přínos pro vývojáře Mapperu by byl v tom, že by mohli jednoduše získat přístup k informacím, kvůli kterým by se museli k danému PC s nainstalovaným Mapperem připojit vzdáleně, případně by museli navštívit čisté prostory. Jedná se například o získávání zá-

znamů z logovacích souborů. Tyto záznamy se totiž zpravidla prohlížejí až ve chvílích, kdy pracovníci zpozorují a nahlásí nějakou chybu. Chyby, jež dokáže vyřešit opětovné spuštění Mapperu se většinou nehlásí a tak zůstávají vývojářům skryty (nemusí ovšem se jednat pouze o chyby v software, ale může být uvolněný kabel spojující PC s testerem).

Pro test inženýry je plánovaná možnost částečného ovládání Mapperu prostřednictvím aplikace Mapper Dashboard. Výhoda je v tom, že pokud si zodpovědný pracovník všimne nějaké chyby při měření, tak jej může rychle zastavit a nemusí volat do čistých prostor. Tím se ušetří hliníkové plošky čipů, které se používají pro testování.

Obě výše zmíněné funkcionality ovšem budou vyžadovat určité prvky zabezpečení (autentizaci a autorizaci uživatele) a bude muset být upraven jak Mapper.NET, tak Mapper Dashboard.

Poslední vylepšení, které bylo pro střednědobý vývoj zvažováno, je propojení Mapper Dashbordu s dalšími aplikacemi. Například s aplikací MapSpy, která umožňuje prohlížet již změřené wafery uložené v databázi. Vize je taková, že uživatel Mapper Dashboardu sledující měření *n-tého* waferu se bude chtít podívat na předcházející, již změřené wafery v daném lotu. Místo otevírání aplikace MapSpy a následného vyhledávání požadovaného lotu by měl uživateli stačit ke stejnému výsledku jediný klik myši, který by toto všechno udělal za něj.

Literatura

[1] WEAVER, James, et al. *Pro JavaFX 2: A Definitive Guide to Rich Clients with Java Technology.* Apress, 2012.

[2] HOXMEIER, John A.; DICESARE, Chris. *System response time and user satisfaction: An experimental study of browser-based applications.* In: Proceedings of the Association of Information Systems Americas Conference. 2000. p. 10-13.

[3] HARRISON, Chris; YEO, Zhiquan; HUDSON, Scott E. *Faster progress bars: manipulating perceived duration with visual augmentations.* In: Proceedings of the 28th international conference on Human factors in computing systems. ACM, 2010. p. 1545-1548.

[4] HARRISON, Chris, et al. *Rethinking the progress bar.* In: Proceedings of the 20th annual ACM symposium on User interface software and technology. ACM, 2007. p. 115-118.

[5] *KOHOUTEK, Rudolf. Vědomosti, dovednosti a návyky žáků (studentů). Psychologie v teorii a praxi* [online]. 2009 [cit. 2013-03-27].
Dostupné z: `http://rudolfkohoutek.blog.cz/0911/vedomosti-dovednosti-a-navyky-zaku-studentu`

[6] *Czochralski process. Wikipedia* [online]. 2013, last modified on 22 February 2013 at 15:21 [cit. 2013-03-27].
Dostupné z: `http://en.wikipedia.org/wiki/Czochralski_process`

[7] *Kde se bere monokrystalický křemík? Tažení monokrystalů křemíku Czochralského metodou. Materiálová věda* [online]. 2011 [cit. 2013-03-27].
Dostupné z: `http://materialovaveda.blogspot.cz/2011/07/kde-se-bere-monokrystalicky-kremik.html`

[8] *Integrovaný obvod. Wikipedia* [online]. 2013, naposledy editována 8. 3. 2013 v 16:34 [cit. 2013-03-27].
Dostupné z: `http://cs.wikipedia.org/wiki/Integrovan%C3%BD_obvod`

[9] KELNAROVÁ, Jarmila a Eva MATĚJKOVÁ. *Psychologie: pro studenty zdravotnických oborů.* 1. vyd. Praha: Grada, 2010, 162 s. Sestra. ISBN 978-802-4732-701.

[10] *Paper prototyping. Wikipedia* [online]. 2013, last modified on 7 March 2013 at 01:20 [cit. 2013-03-28].
Dostupné z: `http://en.wikipedia.org/wiki/Paper_prototyping`

[11] NIELSEN, Jakob. *Response Times: The 3 Important Limits*. Nielsen Norman Group: UX Training: Consulting : Research [online]. © 1998-2013 [cit. 2013-03-28]. Dostupné z: http://www.nngroup.com/articles/response-ti mes-3-important-limits

[12] *User interface: Consistency. Wikipedia* [online]. 2013, last modified on 14 March 2013 at 09:10 [cit. 2013-03-28].
Dostupné z: http://en.wikipedia.org/wiki/User_interface#Consistency

[13] *Windows User Experience Interaction Guidelines* [online]. 2010, Last updated September 29, 2010 [cit. 2013-03-28].
Dostupné z: http://msdn.microsoft.com/en-us/library/windows/desktop/aa511258.aspx

[14] *Colors and the UI: Color Deficiencies*. KOMISCHKE, Tobias. InfoQ [online]. 2008 [cit. 2013-03-28]. Dostupné z: http://www.infoq.com/articles/Colors-UI

[15] *Acceptable Response Times. GNOME Developer Center* [online]. © 2005-2012 [cit. 2013-03-28].
Dostupné z: https://developer.gnome.org/hig-book/3.5/feedback-response-times.html.en

[16] *Dow Corning joins the imec GaN affiliation program. I-Micronews* [online]. 2011 [cit. 2013-03-28].
Dostupné z: http://www.i-micronews.com/lectureArticle.asp?id=6305

Příloha A

Obsah CD

Adresářová struktura CD

- `Data` - složka s daty potřebnými pro off-line spuštění RTM.
  - `probe-area.xml` - mapa Mapperů (matice 3x2).
  - `wafer-data.txt` - záznam měření na Mapperu (11 waferů).
- `Images` - složka s logem Mapper Dashboardu a fotografiemi z čistých prostor.
- `Offline-launch` - složka se soubory potřebnými pro spuštění off-line verze RTM.
- `Poster` - složka obsahující plakát k této práci (velikost A2).
- `Source-code` - zdrojové kódy napsané v rámci bakalářské práce.
  - `Mapper.NET` - složka se zdrojovými kódy části Mapperu, která se stará o odesílání dat na RTM server.
  - `MapperDashboard` - složka obsahující NetBeans projekt pro aplikaci Mapper Dashboard.
  - `MapperEmulator01` - složka obsahující NetBeans projekt pro aplikaci Mapper Emulator.
  - `RtmServer` - složka obsahující NetBeans projekt pro RTM server.
- `Thesis` - složka obsahující zdrojové soubory LaTeXa `Makefile`.
- `Video` - složka s video prezentací a dalšími videi.
  - `RTM-video-presentation.mp4` - video ve formátu `MP4` (rozlišení `1600x1200 px`).

Příloha B

Manuál

Jelikož Mapper Dashboard je částečně závislý na některých online neveřejných službách a zdrojích dat, tak není možno ukázat jeho všechny funkce tak, jak se chovají v reálném prostředí. Off-line verzi si ovšem můžete vyzkoušet na přiloženém CD ve složce `Offline-launch`.

Pro finální vydání vydání této bakalářské práce bylo stahování dat o rozmístění Mapperů nahrazeno off-line načítáním ze souboru `probe-area.xml`. Data ze simulovaného průběhu měření jsou v souboru `wafer-data.txt`.

Postup spuštění

1. **Instalace** - stačí zkopírovat obsah složky `Offline-launch` na pevný disk počítače a mít nainstalovanou JavuFX 2.2 a vyšší.

2. **Spuštění RTM serveru** - jako první by měl být spuštěný server. Ten se dá provést v příkazové řádce příkazem: `$ java -jar ./RealTimeMappingServerV2.jar`, případně přímým spuštěním (pokud máte asociované *.jar soubory s Javou).

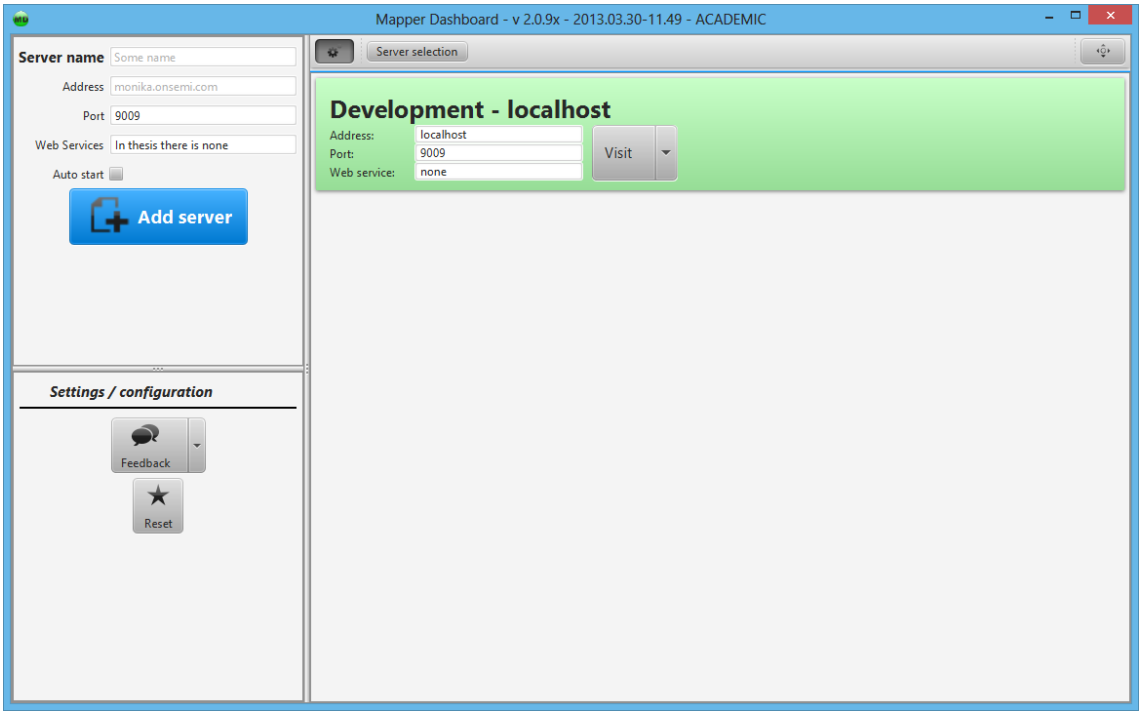
   Takto spuštěný server běží s deseti vlákny v hybridním módu a naslouchá na portu `9009`. Je tedy potřeba zajistit, aby firewall, tento port neblokoval. Pokud vše proběhne správně, měl by se ve stejné složce jako je RTM server, vytvořit logovací soubor `RTM2.log`, do kterého se poté bude zapisovat činnost serveru.

3. **Spuštění Mapper Dashboardu** - ten se spouští souborem `MapperDashboard.jnlp` (tam jsou již nastavené parametry pro off-line načítání dat). Aplikace by po startu měla vypadat tak, jak na obrázku B.1 (A to včetně zeleného pozadí u názvu serveru. Červená by znamenala, že server je off-line a šedá, že je nedostupný.).

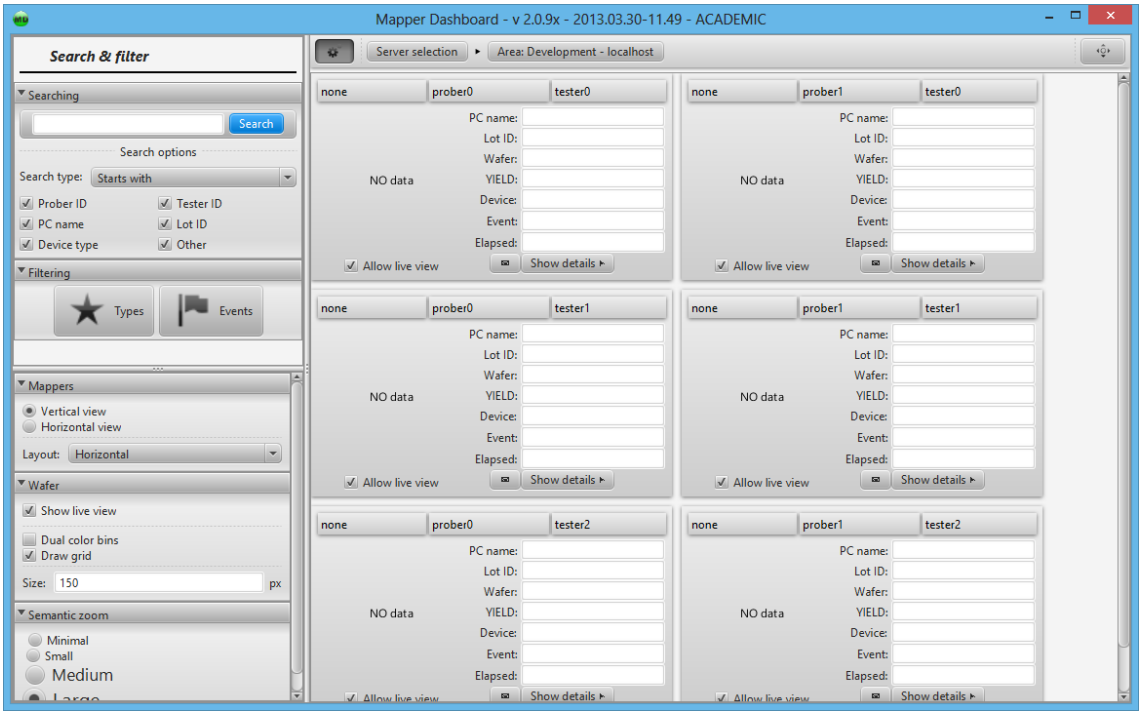
   Po kliknutí na tlačítko *„Visit“*, by se měla zobrazit mapa Mapperů načtená ze souboru `probe-area.xml` a výsledek by měl vypadat jako na obrázku B.2.

   Místo vykreslených waferů ovšem nápis *„NO data“*. Jak zasílat data na RTM server bude vysvětleno v dalším kroku.

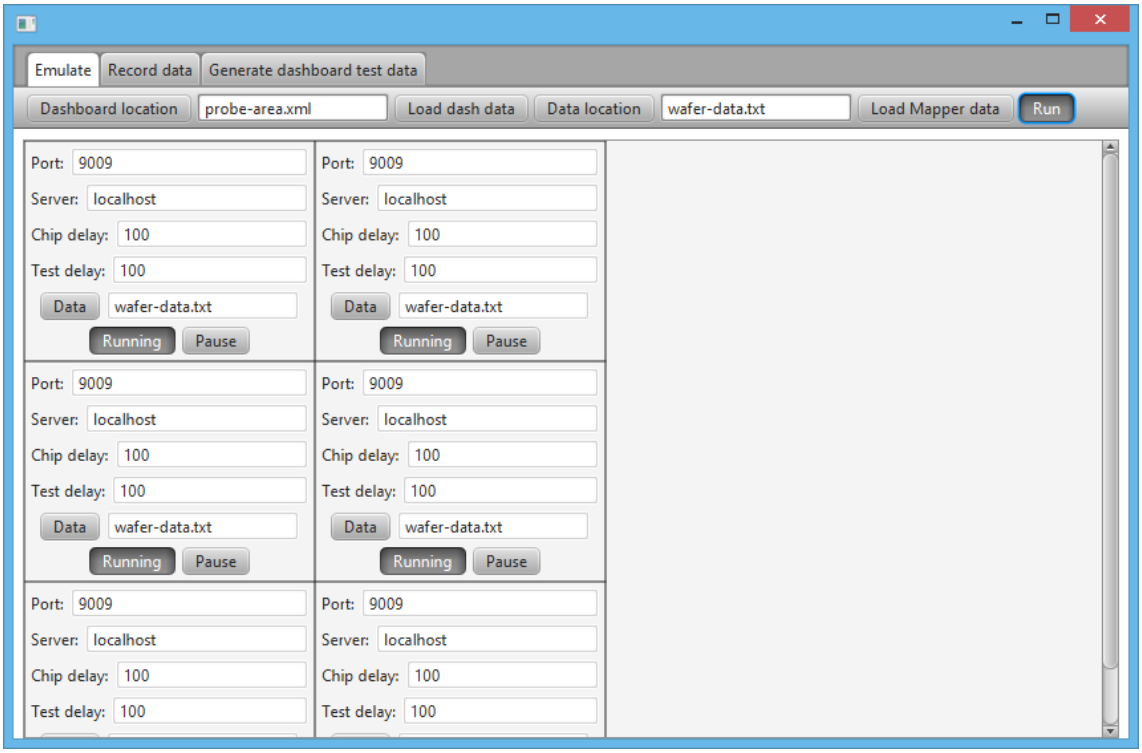
4. **Spuštění Mapper Emulatoru** - Mapper Emulator je aplikace vytvořená k simulacím velkého počtu připojených Mapperů (mimo jiné se chová jako odlehčený RTM server a dokáže získávat data o měření z Mapperů a generovat maticové mapy). Tato aplikace má už v záložce *„Emulate“* nastavené cesty ke zdrojům dat. Jediné co stačí udělat, je: kliknout na tlačítko *„Load dash data“*, poté na *„Load Mapper Data“* a nakonec na *„Run“* (tím se započne odesílání dat na lokální RTM server).



Obrázek B.1: Off-line verze Mapper Dashboardu při prvním spuštění.

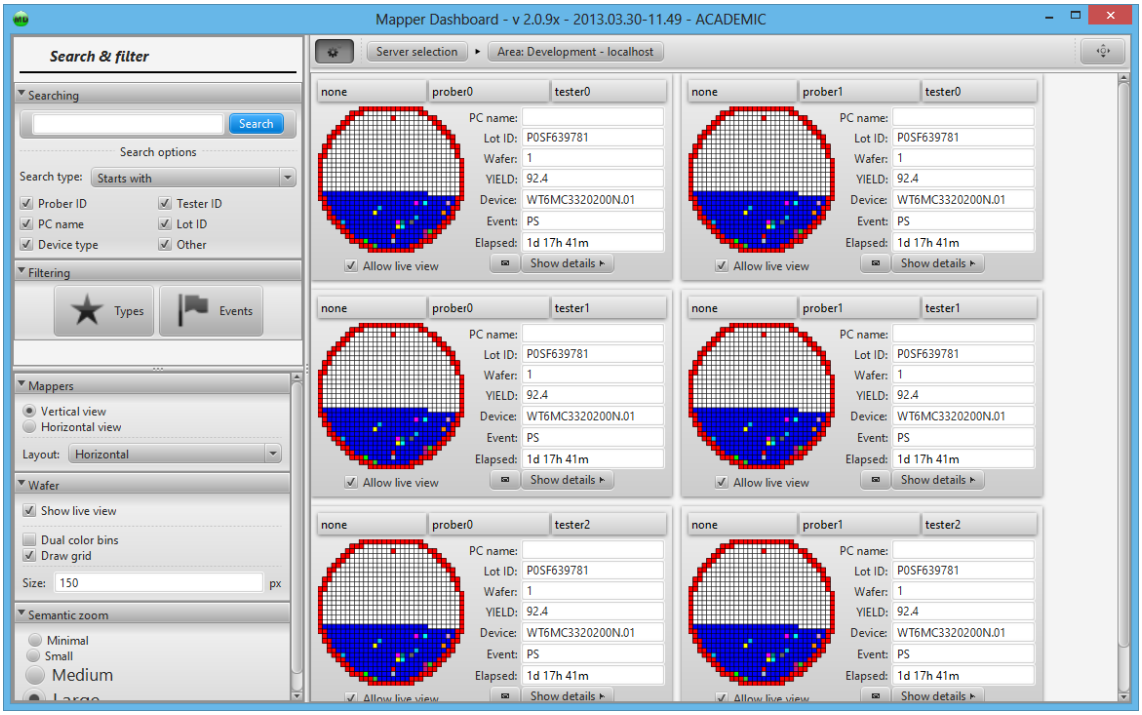


Obrázek B.2: Off-line verze Mapper Dashboardu při načtení mapy Mapperů.

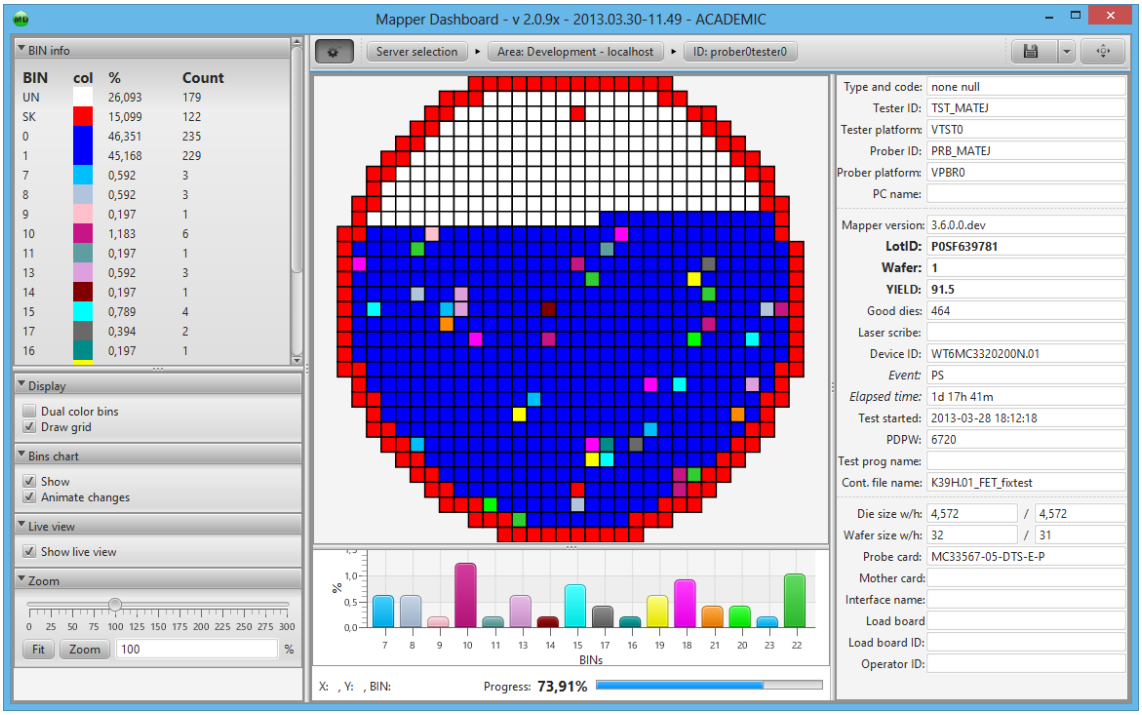


Obrázek B.3: Simulace Mapperů pomoci Mapper Emulatoru.

Po provedení těchto kroků Mapper Emulator vypadá jako na obrázku B.3 a v Mapper Dashboardu se začne ukazovat průběh měření (viz obrázky B.4 a B.5).



Obrázek B.4: Off-line verze Mapper Dashboardu při vykreslování dat z RTM serveru.



Obrázek B.5: Off-line verze Mapper Dashboardu v detailním náhledu na Mapper.

Příloha C

Logo Mapper Dashboardu

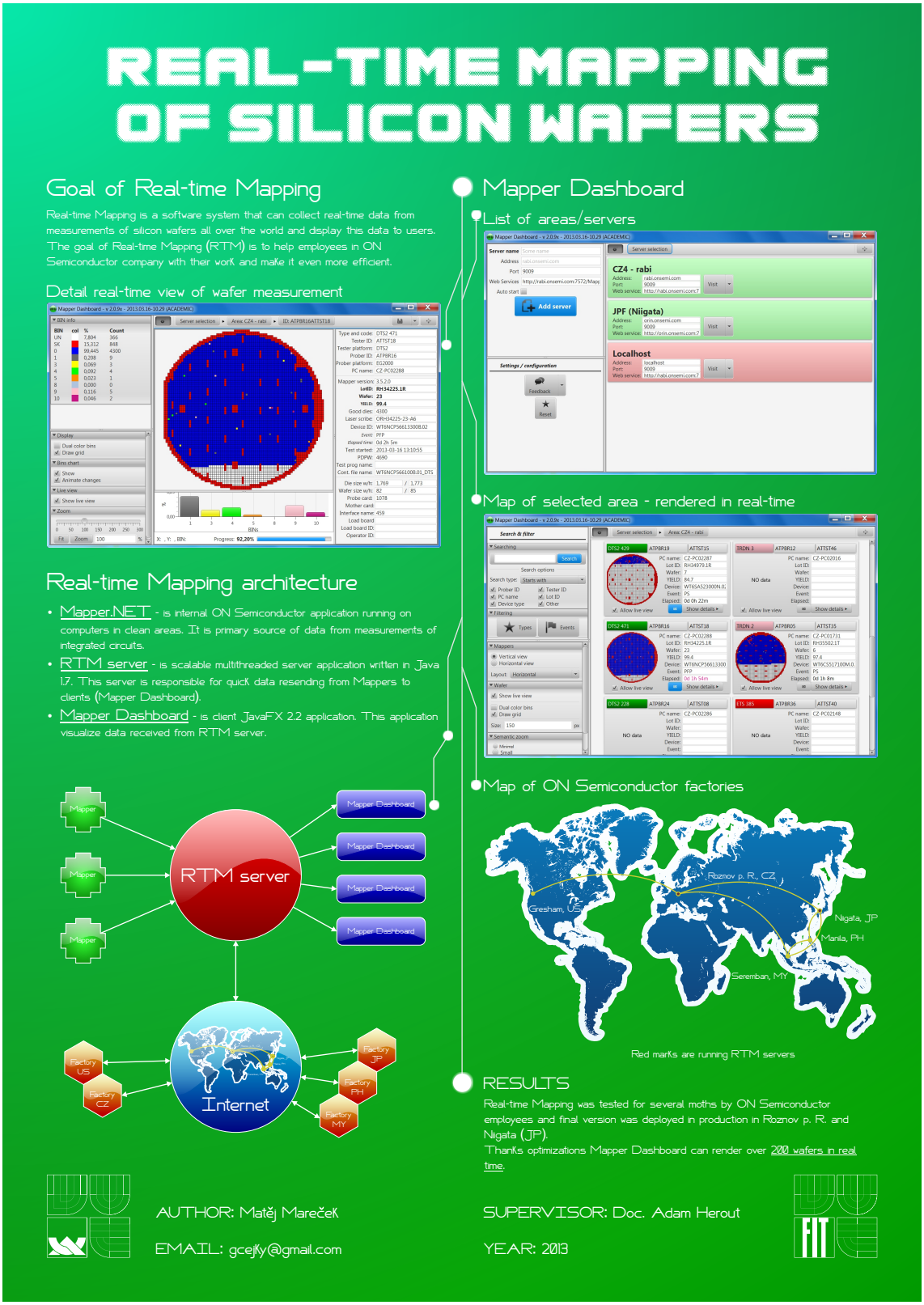
Obrázek C.1: Logo vytvořené pro Mapper Dashboard.

Příloha D

Plakát

Informace

- **Velikost** - A2 (420 x 594 mm)
- **Jazyk** - angličtina
- **Umístění na CD** - `Poster/RTM-poster.pdf`



Obrázek D.1: Plakát sloužící k prezentaci této bakalářské práce.